About Our Cover:

The design on our front cover is an abstract portrayal of the basic microcontroller function. The center sphere, symbolic of a microcontroller, contains a molecular orbital diagram of the architectural construction of a cubic unit of silicon. The red pathways leading from the central sphere, are symbolic of distant or remote controlled applications.
Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, I, ICE, ICS, iDBP, iDIS, iICE, iLBX, iMMX, Insite, Intel, intIBOS, Intelevention, intelligent Identifier, intelligent Programming, Intellec, Intellink, iOSP, iPDS, iSBC, iSBX, iSDM, iSXm, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, Plug-A-Bubble, PROMPT, Promware, QUEST, QUEX, Ripplemode, RMX/80, RUPI, Seamless, SOLO, SYSTEM 2000, and UPI, and the combination of ICE, ICS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051

©INTEL CORPORATION 1984
# Table of Contents

**ALPHANUMERIC INDEX** ................................................................. iii

**MCS®-96 FAMILY**

**CHAPTER 1**
- Introduction To MCS®-96 ......................................................... 1-1

**CHAPTER 2**
- Architectural Overview .......................................................... 2-1

**CHAPTER 3**
- MCS®-96 Software Design Information ........................................ 3-1

**CHAPTER 4**
- MCS®-96 Hardware Design Information .......................................... 4-1

**CHAPTER 5**
- MCS®-96 Data Sheet ................................................................. 5-1

**CHAPTER 6**
- MCS®-96 Article Reprint
  AR-321: High Performance Event Interface For A Microcomputer .......... 6-1

**MCS®-51 FAMILY**

**CHAPTER 7**
- MCS®-51 Architecture ............................................................. 7-1

**CHAPTER 8**
- MCS®-51 Instruction Set ........................................................... 8-1

**CHAPTER 9**
- MCS®-51 Data Sheets
  8031/8051 8031AH/8051AH 8032AH/8052AH 8751H/8751H-12 ................. 9-1
  8052AH-Basic ............................................................................. 9-15
  8031AH/8051AH 8032AH/8052AH 8751H/8751H Express ....................... 9-39

**CHAPTER 10**
- MCS®-51 Application Notes
  AP-69: An Introduction To The Intel MCS®-51 Single-Chip Microcomputer Family .......................................................... 10-1
  AP-70: Using The Intel MCS®-51 Boolean Processing Capabilities ........ 10-31
  AP-223: 8051 Based CRT/Terminal Controller ................................ 10-65

**CHAPTER 11**
- MCS®-51 Article Reprint
  AR-224: Controller Chip Takes On Many Industrial, Computer Uses ........ 11-1

**MCS®-48 FAMILY**

**CHAPTER 12**
- MCS®-48 Single Component System ............................................. 12-1

**CHAPTER 13**
- MCS®-48 Expanded System ......................................................... 13-1

**CHAPTER 14**
- MCS®-48 Instruction Set ............................................................. 14-1

**CHAPTER 15**
- MCS®-48 Data Sheets
  8243 ....................................................................................... 15-1
  8048AH/8035AH/8049AH/8039AH/8050AH/8040AH ......................... 15-7
  8748H/8035H/8749H/8039H ......................................................... 15-17
  MCS®-48 Express ....................................................................... 15-30
  80C39-9/80C49-7 ................................................................. 15-34
THE RUPI™ FAMILY: MICROCONTROLLER
WITH ON-CHIP COMMUNICATION CONTROLLER

CHAPTER 16
The RUPI™-44 Family .............................................. 16-1

CHAPTER 17
8044 Architecture .............................................. 17-1

CHAPTER 18
8044 Serial Interface .............................................. 18-1

CHAPTER 19
8044 Application Examples .............................................. 19-1

CHAPTER 20
RUPI™ Data Sheets
8044AH/8344AH .............................................. 20-1
8744 .............................................. 20-20

CHAPTER 21
RUPI™ Article Reprint .............................................. 21-1

DESIGN CONSIDERATIONS

CHAPTER 22
Application Notes
AP-125: Designing Microcontroller Systems
For Electrically Noisy Environments .............................................. 22-1
AP-155: Oscillators For Microcontrollers .............................................. 22-23

DESIGN CONSIDERATIONS
WHEN USING CHMOS

CHAPTER 23 .............................................. 23-1
Article Reprints
AR-302 .............................................. 23-6
AR-332 .............................................. 23-17

ADVANCED PACKAGING INFORMATION

CHAPTER 24 .............................................. 24-1
### ALPHANUMERICAL INDEX

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8031</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>8031AH</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>8031AH</td>
<td>Express Data Sheet</td>
<td>9-39</td>
</tr>
<tr>
<td>8032AH</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>8032AH</td>
<td>Data Sheet</td>
<td>9-39</td>
</tr>
<tr>
<td>8035H</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8035AHL</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8039H</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8039AHL</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8040AHL</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8044</td>
<td>Application Example</td>
<td>19-1</td>
</tr>
<tr>
<td>8044</td>
<td>Architecture</td>
<td>17-1</td>
</tr>
<tr>
<td>8044</td>
<td>Serial Interface</td>
<td>18-1</td>
</tr>
<tr>
<td>8044AH</td>
<td>Data Sheet</td>
<td>20-1</td>
</tr>
<tr>
<td>8048AH</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8049AH</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8050AH</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8051</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>8051AH</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>8051AH</td>
<td>Express Data Sheet</td>
<td>9-39</td>
</tr>
<tr>
<td>8052AH</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>8052AH</td>
<td>Basic Data Sheet</td>
<td>9-15</td>
</tr>
<tr>
<td>8052AH</td>
<td>Express Data Sheet</td>
<td>9-39</td>
</tr>
<tr>
<td>8243</td>
<td>Data Sheet</td>
<td>15-1</td>
</tr>
<tr>
<td>8344AH</td>
<td>Data Sheet</td>
<td>20-1</td>
</tr>
<tr>
<td>8744</td>
<td>Data Sheet</td>
<td>20-20</td>
</tr>
<tr>
<td>8748H</td>
<td>Data Sheet</td>
<td>15-17</td>
</tr>
<tr>
<td>8749H</td>
<td>Data Sheet</td>
<td>15-7</td>
</tr>
<tr>
<td>8751H</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>8751H</td>
<td>Express Data Sheet</td>
<td>9-39</td>
</tr>
<tr>
<td>8751H-12</td>
<td>Data Sheet</td>
<td>9-1</td>
</tr>
<tr>
<td>80C31BH</td>
<td>Data Sheet</td>
<td>9-24</td>
</tr>
<tr>
<td>80C31BH-2</td>
<td>Data Sheet</td>
<td>9-24</td>
</tr>
<tr>
<td>80C39-9</td>
<td>Data Sheet</td>
<td>15-34</td>
</tr>
<tr>
<td>80C51BH</td>
<td>Data Sheet</td>
<td>9-24</td>
</tr>
<tr>
<td>80C51BH-2</td>
<td>Data Sheet</td>
<td>9-24</td>
</tr>
<tr>
<td>ADVANCED Packaging Information</td>
<td>24-1</td>
<td></td>
</tr>
<tr>
<td>Design Considerations</td>
<td>22-1</td>
<td></td>
</tr>
</tbody>
</table>
# ALPHANUMERICAL INDEX

Design Considerations Application Notes ........................................ 22-2, 22-24
Design Considerations When Using CHMOS ........................................ 23-1
Design Considerations When Using CHMOS Article Reprints ................... 23-6, 23-17
MCS-48 Data Sheets .......................................................... 15-1, 15-7, 15-17, 15-30, 15-34
MCS-48 Expanded System .......................................................... 13-1
MCS-48 Instruction Set .................................................................. 14-1
MCS-48 Single Component System .................................................. 12-1
MCS-51 Application Notes ......................................................... 10-1, 10-31, 10-65
MCS-51 Architecture .................................................................. 7-1
MCS-51 Article Reprint .................................................................. 11-1
MCS-51 Data Sheets .................................................................... 9-1, 9-15, 9-24, 9-39
MCS-51 Instruction Set ............................................................... 8-1
MCS-96 Architectural Overview .................................................... 2-1
MCS-96 Article Reprint .................................................................. 6-1
MCS-96 Data Sheet ....................................................................... 5-1
MCS-96 Hardware Information ....................................................... 4-1
MCS-96 Introduction .................................................................... 1-1
MSC-96 Software Design Information .............................................. 3-1
RUPI Data Sheets ........................................................................ 20-1, 20-20
RUPI Article Reprints ................................................................. 21-1
RUPI-44 Family ........................................................................... 16-1
1.0 CONTINUING MICROCONTROLLER EVOLUTION

Beginning with the introduction of the world standard 8048 (MCS®-48) Microcontroller in 1976, Intel has continued to drive the evolution of single chip microcontrollers. In 1980, Intel introduced the 8051 (MCS-51) offering performance levels significantly higher than the 8048. With the advent of the 8051, the microcontroller applications base took a marked vertical leap. These versatile chips are used in applications from keyboards and terminals to controlling automobile engines. The 8051 quickly gained the position of the second generation world standard microcontroller.

Now that the semiconductor process technologies are being pushed to new limits, it has become possible to integrate more than 100,000 transistors onto a single silicon chip. Microcontroller designers at Intel have taken today's process technology achievements and forged a new generation of single chip microcontrollers called the MCS-96. The 8096 (generic part number for MCS-96) offers the highest level of system integration ever achieved on a single chip microcontroller. It uses over 120,000 transistors to implement a high performance 16-bit CPU, 8K bytes of program memory, 232 bytes of data memory and both analog and digital types of I/O features. Figure 1-1 shows the evolution of single chip microcontroller at Intel.

![Figure 1-1. Evolution of Microcontrollers at Intel](image-url)
1.1 INTRODUCTION TO THE MCS®-96

The 8096 consists of a 16-bit powerful CPU tightly coupled with program and data memory along with several I/O features all integrated onto a single piece of silicon. The CPU supports bit, byte, and word operations. 32-bit double words are also supported for a subset of the instruction set. With a 12 MHz input frequency, the 8096 can perform a 16-bit addition in 1.0 μs and 16 × 16 multiply or 32/16 divide in 6.5 μs.

Four high-speed trigger inputs are provided to record the times at which external events occur with a resolution of 2 μs (at 12 MHz crystal frequency). Up to six high-speed pulse generator outputs are provided to trigger external events at preset times. The high speed output unit can simultaneously perform timer functions, up to four such 16-bit software timers can be in operation at once in addition to the two 16-bit hardware timers.

An optional on-chip A/D converter converts up to four (in the 48-pin version) or 8 (in the 68-pin version) analog input channels into 10-bit digital values. Also provided on-chip, is a serial port, a watchdog timer, and a pulse-width modulated output signal. Table 1.1 shows the features and benefits summary for the MCS-96.

The 8096 with its 16-bit CPU and all the I/O features and interface resources on a single piece of silicon represents the highest level of system integration in the world of microcontrollers. It will open up new applications which had to use multiple chip solutions in the past.

<table>
<thead>
<tr>
<th>FEATURES</th>
<th>BENEFITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-Bit CPU</td>
<td>Efficient machine with higher throughput.</td>
</tr>
<tr>
<td>8K Bytes ROM</td>
<td>Large program space for more complex, larger programs.</td>
</tr>
<tr>
<td>232 Bytes RAM</td>
<td>Large on-board register file.</td>
</tr>
<tr>
<td>Hardware MUL/DIV</td>
<td>Provides good math capability 16 by 16 multiply or 32 by 16 divide in 6.5 μs @ 12 MHz.</td>
</tr>
<tr>
<td>6 Addressing Modes</td>
<td>Provides greater flexibility of programming and data manipulation.</td>
</tr>
<tr>
<td>High Speed I/O Unit</td>
<td>Can measure and generate pulses with high resolution (2 μs @ 12 MHz).</td>
</tr>
<tr>
<td>4 dedicated I/O lines</td>
<td></td>
</tr>
<tr>
<td>4 programmable I/O lines</td>
<td></td>
</tr>
<tr>
<td>10-Bit A/D Converter</td>
<td>Reads the external analog inputs.</td>
</tr>
<tr>
<td>Full Duplex Serial Port</td>
<td>Provides asynchronous serial link to other processors or systems.</td>
</tr>
<tr>
<td>Up to 40 I/O Ports</td>
<td>Provides TTL compatible digital data I/O including system expansion with standard 8 or 16-bit peripherals.</td>
</tr>
<tr>
<td>Programmable 8 Source Priority Interrupt System</td>
<td>Respond to asynchronous events.</td>
</tr>
<tr>
<td>Pulse Width Modulated Output</td>
<td>Provides a programmable pulse train with variable duty cycle. Also used to generate analog output.</td>
</tr>
<tr>
<td>Watchdog Timer</td>
<td>Provides ability to recover from software malfunction or hardware upset.</td>
</tr>
<tr>
<td>48 Pin (DIP) &amp; 68 Pin (Flatpack, Pin Grid Array) Versions</td>
<td>Offers a variety of package types to choose from to better fit a specific application need for number of I/O’s and package size.</td>
</tr>
</tbody>
</table>
1.2. MCS®-96 APPLICATIONS

The MCS-96 products are stand-alone high performance single chip microcontrollers designed for use in sophisticated real-time demanding applications such as industrial control, instrumentation and intelligent computer peripherals. The wide base of applications cut across all industry segments (see table 1.2). With the 16-bit CPU horsepower, high-speed math processing and high-speed I/O, the 8096 is ideal for complex motor control and axis control systems. Examples include three phase, large horsepower AC motors and robotics.

With its 10-bit A/D converter option, the device finds usage in data acquisition systems and closed-loop analog controllers. It permits considerable system integration by combining analog and digital I/O processing in the single chip.

This chip is ideally suited in the area of instrumentation products such as gas chromatographs, which combine analog processing with high speed number crunching. The same features make it a desirable component for aerospace applications like missile guidance and control.

1.3. MCS®-96 FAMILY DEVELOPMENT SUPPORT TOOLS

The product family is supported by a range of Intel software and hardware development tools. These tools shorten the product development cycle, thus bringing the product to the market sooner.

1.3.1. MCS®-96 Software Development Package

The 8096 software development package provides development system support specifically designed for the MCS-96 family of single chip microcontrollers. The package consists of a symbolic macro assembler ASM-96, Linker/Relocator RL-96 and the librarian LIB-96. Among the high level languages, PLM-96 is offered along with a floating point math package. Additional high level languages are being developed for the MCS-96 product family.

1.3.2. ASM-96 MACRO Assembler

The 8096 macro assembler translates the symbolic assembly language instructions into the machine executable object code. ASM-96 enables the programmer to write the program in a modular fashion. The modular programs divide a rather complex program into smaller functional units, that are easier to code, to debug, and to change. The separate modules can then be linked and located into one program module using the RL-96 utility. This utility combines the selected input object modules into a single output object module. It also allocates memory to input segments and binds the relocatable addresses to absolute addresses. It then produces a print file that consists of a link summary, a symbol table listing and an intermediate cross-reference listing. LIB-96, another utility helps to create, modify, and examine library files. The ASM-96 runs on Intellec Series III or IV.

1.3.3. PL/M-96

The PL/M-96 compiler translates the PL/M-96 language into 8096 relocatable object modules. This allows improved programmer productivity and application reliability. This high level language has been efficiently designed to map into the machine architecture, so as not to trade off higher programmer productivity with inefficient code. Since the language and the compiler are optimized for the 8096 and its application environment, developing software with PL/M-96 is a ‘low-risk’ project.

---

Table 1-2 MCS®-96 Broad Base of Applications

<table>
<thead>
<tr>
<th>INDUSTRIAL</th>
<th>MOTOR CONTROL</th>
<th>ROBOTICS</th>
<th>DISCRETE AND CONTINUOUS PROCESS CONTROL</th>
<th>NUMERICAL CONTROL</th>
<th>INTELLIGENT TRANSDUCERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTRUMENTATION</td>
<td>MEDICAL INSTRUMENTATION</td>
<td>LIQUID AND GAS CHROMATOGRAPHS</td>
<td>OSCILLOSCOPES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSUMER</td>
<td>VIDEO RECORDER</td>
<td>LASER DISK DRIVE</td>
<td>HIGH-END VIDEO GAMES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUIDANCE &amp; CONTROL</td>
<td>MISSILE CONTROL</td>
<td>TORPEDO GUIDANCE CONTROL</td>
<td>INTELLIGENT AMMUNITION</td>
<td>AEROSPACE GUIDANCE SYSTEMS</td>
<td></td>
</tr>
<tr>
<td>DATA PROCESSING</td>
<td>PLOTTERS</td>
<td>COLOR AND B&amp;W COPIERS</td>
<td>WINCHESTER DISK DRIVE</td>
<td>TAPE DRIVES</td>
<td>IMPACT AND NON-IMPACT PRINTERS</td>
</tr>
<tr>
<td>AUTOMOTIVE</td>
<td>ILLUMINATION CONTROL</td>
<td>TRANSMISSION CONTROL</td>
<td>ANTI SKID BRAKING</td>
<td>EMISSION CONTROL</td>
<td></td>
</tr>
</tbody>
</table>

---

1-3
INTRODUCTION TO MCS®-96

1.3.4. Hardware Development Support: iSBE-96
The iSBE-96 is a hardware execution and debug tool for the MCS-96 products. It consists of a monitor/debugger resident in an 8096 system. This development system interfaces with the user’s 8096 system via two ribbon cables, one for the 8096 I/O ports, and the other for the memory bus. The iSBE-96 is controlled by an Intellec Series III or other computer system over a serial link. Power for the iSBE-96 can be supplied by plugging it into the MULTIBUS® card slot, or by an external power supply. The iSBE-96 is contained on one standard MULTIBUS board.

The iSBE-96 provides the most often used features for real-time hardware emulation. The user can display and modify memory, set up break points, execute with or without breakpoints and change the memory map. In addition, the user can single step through the system program.

1.3.5. MCS®-96 Workshop
The workshop provides the design engineer or system designer hands-on experience with the MCS-96 family of products. The course includes an explanation of the Intel 8096 architecture, system timing, input/output design. The lab sessions allow the attendees to gain in-depth knowledge of the MCS-96 product family and support tools.

1.3.6. Insite™ Library
The Intel Insite Library contains several application programs. A very useful program contained in the Insite is SIM-96, the software simulator for 8096. It allows software simulations of user’s system. The simulator provides the ability to set breakpoints, examine and modify memory, disassemble the object code and single step through the code.

1.4. MCS®-96 FAMILY OF PRODUCTS
Although 8096 is the generic part number often used for the MCS-96 products throughout this manual, the product family consists of eight configurations with eight part numbers including the 8096. This wide variety of products is offered to best meet user’s application requirements in terms of number of I/O’s and package size. The options include on-board 8K bytes of mask programmed memory, 10-bit A/D converter, and 48 or 68 pin package type.

Table 1-3 summarizes all the current products in the MCS®-96 product family.

<table>
<thead>
<tr>
<th>OPTIONS</th>
<th>68 PIN</th>
<th>48 PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGITAL I/O</td>
<td>ROMLESS</td>
<td>8096</td>
</tr>
<tr>
<td></td>
<td>ROM</td>
<td>8396</td>
</tr>
<tr>
<td>ANALOG AND DIGITAL I/O</td>
<td>ROMLESS</td>
<td>8097</td>
</tr>
<tr>
<td></td>
<td>ROM</td>
<td>8397</td>
</tr>
</tbody>
</table>

The 48 pin version is available in a DIP (dual inline) package.

The 68 pin version comes in two packages, the Plastic Flatpack and the Pin Grid Array.
CHAPTER 2
ARCHITECTURAL OVERVIEW

2.0. INTRODUCTION

The 8096 can be separated into several sections for the purpose of describing its operation. There is a CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator and the back-bias generator. The CPU and the programmable I/O make the 8096 very different from any other microcontroller, let us first examine the CPU.

2.1. CPU OPERATION

The major components of the CPU on the 8096 are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

2.1.1. CPU Buses

A "Control Unit" and two buses connect the Register File and RALU. Figure 2-1 shows the CPU with its major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to

Figure 2-1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)
the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

2.1.2. CPU Register File
The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

2.1.3. RALU Control
Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 2-1 shows the instruction register and the control unit.

2.1.4. RALU
Most calculations performed by the 8096 take place in the RALU. The RALU, shown in Figure 2-2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16 + sign extension) wide. Some of the registers have the ability to perform simple operations to off-load the ALU.

A separate incrementer is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when double-word quantities are being shifted, the "Upper Word" register is used

---

**Figure 2-2. RALU Block Diagram**

2-2
whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

The DELAY shown in Figure 2-2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

2.2. BASIC TIMING

The 8096 requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 2-3. Details of the circuit and suggestions for its use can be found in section 4.1.

2.2.1. Internal Timings
The crystal or external oscillator frequency is divided by 3 to generate the three internal timing phases as shown in Figure 2-4. Each of the internal phases repeat every 3 oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096 operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 2-4. It should be noted that propagation delays have not
been taken into account in this diagram. Details on these and other timing relationships can be found in sections 4.1, 4.4 and 4.6.

The RESET line can be used to start the 8096 at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under RESET, sections 2.15 and 4.1.

2.3. MEMORY SPACE

The addressable memory space on the 8096 consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2010H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in figure 2-5.

2.3.1. Register File

Locations 00H through 0FFH contain the Register File and SFRs. Complete information on this section of memory space can be found in section 2.4. No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 0000H through 0FFH is made, the instructions will be fetched from external memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location 0000H, therefore, the NMI instruction is also reserved for Intel development tools.

2.3.2. Reserved Memory Spaces

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 4.6.7. If ports 3 and 4 are not going to be reconstructed then these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in section 2.5. Internal locations 2012H through 207FH are reserved for Intel’s factory test code. To ensure compatibility with future parts external locations 2012H through 207FH must contain the hex value FFH.

Resetting the 8096 causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in section 2.15.

2.3.3. Internal ROM

When a ROM part is ordered, the internal memory locations 2080H through 3FFFH are user specified as are the interrupt vectors in locations 2000H through 2011H.

---

**Figure 2-5. Memory Map**
Instruction and data fetches from the internal ROM occur only if the part has a ROM, EA is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory.

2.3.4. Memory Controller
The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-bus and several control lines. Since the A-bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 3 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Tables 3-3 and 3-4 show the normal execution times with no wait states added. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in Table 3-4.

2.3.5. System Bus
External memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in section 4.6. Since the 8096's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable (BHE) and Address/Data Line 0 (AD0). The BHE line must be transparently latched, just as the addresses are.

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).
When BHE is active (low), the memory connected to the high byte of the data bus should be selected. When MAO is low the memory connected to the low byte of the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only (MAO = 0, BHE = 1), to the high (odd) byte only (MAO = 1, BHE = 0), or to both bytes (MAO = 0, BHE = 0). When a memory block is being used only for reads, BHE and MAO need not be decoded.

Figure 2-6 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on AD0-AD15 and BHE is set to the required state. ALE then falls, the address is taken off the pins, and the RD (Read) signal goes low. The READY line can be pulled low to hold the processor in this condition for a few extra state times.

2.3.6. Bus Control Lines

The READY line can be used to hold the processor in the above condition in order to allow access to slow memories or for DMA purposes. Sampling of the READY line occurs internally during Phase A, which is the signal that generates CLKOUT. There is a minimum time in which READY must be stable before CLKOUT goes low. If this set-up time is violated while the part is going to the not-ready state, the part may fail to operate predictably.

Since READY is synchronized with CLKOUT, the 8096 will be in a not-ready condition for a period equal to some multiple of CLKOUT, although the READY line can be brought high at any time. There is a maximum time for holding the 8096 in the not-ready condition, typically on the order of 1 μS. The exact time is specified in the data sheet for the particular part and temperature range desired.

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of RD. The rising edge of RD latches the information into the 8096. If the read is for data, the INST pin will be low when the address is valid, if it is for an instruction the INST pin will be high during this time. The 48-lead part does not have the INST pin.

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write (WR) signal is used instead of the RD signal. The timings are the same until the falling edge of the WR line. At this point the 8096 removes the address and places the data on the bus. The READY line must be held in the desired state at that time as described above. When the WR line goes high the data should be latched to the external memory. INST is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.

Figure 2-7. Register File Memory Map

2.4. RAM SPACE

The internal register locations on the 8096 are divided into two groups, a register file and a set of Special Function Registers (SFRs). The RALU can operate on any of these 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. The stack pointer must be initialized by the user program and can point anywhere in the 64K memory space. The stack builds down. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.
2.4.1. Special Function Registers
All of the I/O on the 8096 is controlled through the SFRs. Many of these registers serve two functions; one if they are read from, the other if they are written to. Figure 2-7 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown.

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.</td>
<td>3.2.7</td>
</tr>
<tr>
<td>AD _ RESULT</td>
<td>A/D Result Hi/Low — Low and high order Results of the A/D converter (byte read only)</td>
<td>2.9.3</td>
</tr>
<tr>
<td>AD _ COMMAND</td>
<td>A/D Command Register — Controls the A/D</td>
<td>2.9.2</td>
</tr>
<tr>
<td>HSI _ MODE</td>
<td>HSI Mode Register — Sets the mode of the High Speed Input unit.</td>
<td>2.7.1</td>
</tr>
<tr>
<td>HSI _ TIME</td>
<td>HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered. (word read only)</td>
<td>2.7.4</td>
</tr>
<tr>
<td>HSO _ TIME</td>
<td>HSO Time Hi/Lo — Sets the time for the High Speed Output to execute the command in the Command Register. (word write only)</td>
<td>2.8.3</td>
</tr>
<tr>
<td>HSO _ COMMAND</td>
<td>HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.</td>
<td>2.8.2</td>
</tr>
<tr>
<td>HSI _ STATUS</td>
<td>HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers.</td>
<td>2.7.4</td>
</tr>
<tr>
<td>SBUF (TX)</td>
<td>Transmit buffer for the serial port, holds contents to be outputed.</td>
<td>2.11</td>
</tr>
<tr>
<td>SBUF (RX)</td>
<td>Receive buffer for the serial port, holds the byte just received by the serial port.</td>
<td>2.11</td>
</tr>
<tr>
<td>INT _ MASK</td>
<td>Interrupt Mask Register — Enables or disables the individual interrupts.</td>
<td>2.5.2</td>
</tr>
<tr>
<td>INT _ PENDING</td>
<td>Interrupt Pending Register — Indicates when an interrupt signal has occurred on one of the sources.</td>
<td>3.6.2</td>
</tr>
<tr>
<td>WATCHDOG</td>
<td>Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.</td>
<td>2.14</td>
</tr>
<tr>
<td>TIMER1</td>
<td>Timer 1 Hi/Lo — Timer 1 high and low bytes. (word read only)</td>
<td>2.6.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.7-8</td>
</tr>
<tr>
<td>TIMER2</td>
<td>Timer 2 Hi/Lo — Timer 2 high and low bytes. (word read only)</td>
<td>2.6.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.7-8</td>
</tr>
<tr>
<td>IOPORT0</td>
<td>Port 0 Register — Levels on pins of port 0.</td>
<td>2.12.1</td>
</tr>
<tr>
<td>BAUD _ RATE</td>
<td>Register which contains the baud rate, this register is loaded sequentially.</td>
<td>2.11.4</td>
</tr>
<tr>
<td>IOPORT1</td>
<td>Port 1 Register — Used to read or write to Port 1.</td>
<td>2.12.2</td>
</tr>
<tr>
<td>IOPORT2</td>
<td>Port 2 Register — Used to read or write to Port 2.</td>
<td>2.12.3</td>
</tr>
<tr>
<td>SP _ STAT</td>
<td>Serial Port Status — Indicates the status of the serial port.</td>
<td>2.11.3</td>
</tr>
<tr>
<td>SP _ CON</td>
<td>Serial port control — Used to set the mode of the serial port.</td>
<td>2.11.1</td>
</tr>
<tr>
<td>IOS0</td>
<td>I/O Status Register 0 — Contains information on the HSO status.</td>
<td>2.13.4</td>
</tr>
<tr>
<td>IOS1</td>
<td>I/O Status Register 1 — Contains information on the status of the timers and of the HSI.</td>
<td>2.13.5</td>
</tr>
<tr>
<td>IOC0</td>
<td>I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.</td>
<td>3.7.2</td>
</tr>
<tr>
<td>IOC1</td>
<td>I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.</td>
<td>2.13.3</td>
</tr>
<tr>
<td>PWM _ CONTROL</td>
<td>Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.</td>
<td>2.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.3.2</td>
</tr>
<tr>
<td>OPORT2</td>
<td>Port 2 Register — Used to read or write to Port 2.</td>
<td>2.12.3</td>
</tr>
</tbody>
</table>

Figure 2-8. SFR Summary
in Figure 2-8, with complete descriptions reserved for later chapters. Note that these registers can be accessed only as bytes unless otherwise indicated.

Within the SFR space are several registers labeled as "RE­SERVED". These registers are reserved for future expansion or test purposes. Reads or writes of these registers may produce unexpected results. For example, writing to location 000CH will set both timers to 0FFFXH, this feature is for use in testing the part and should not be used in programs.

2.4.2. Power Down
The upper 16 RAM locations (0FOH through 0FFH) receive their power from both the VCC pin and the VPD pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the VPD pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification).

To place the 8096 into a power down mode, the RESET pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the VCC pin, the VPD pin must remain within specifications. The 8096 can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096 out of power down, RESET is held low while VCC is applied. Two state times after the oscillator and the back bias generator have stabilized (~1 millisecond), the RESET pin can be pulled high. The 8096 will begin to execute code at location 02080H 10 state times after RESET is pulled high. Figure 2-9 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1 cycles be used. Suggestions for actual hardware connections are given in section 4.1. Reset is discussed in section 2.15.

2.5. INTERRUPT STRUCTURE

2.5.1. Interrupt Sources
Eight interrupt sources are available on the 8096. When enabled, an interrupt occurring on any of these sources will force a call to the location stored in the vector location for that source. The interrupt sources and their respective vector locations are listed in Figure 2-10. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use by Intel development systems. Many of the interrupt sources can be activated by several methods, Figure 2-11 shows all of the possible sources for interrupts.

<table>
<thead>
<tr>
<th>Source</th>
<th>Vector Location (High Byte)</th>
<th>Vector Location (Low Byte)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>2011H</td>
<td>2010H</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Extint</td>
<td>200FH</td>
<td>200EH</td>
<td>7 (Highest)</td>
</tr>
<tr>
<td>Serial Port</td>
<td>200DH</td>
<td>200CH</td>
<td>6</td>
</tr>
<tr>
<td>Software Timers</td>
<td>200BH</td>
<td>200AH</td>
<td>5</td>
</tr>
<tr>
<td>HSI.0</td>
<td>2009H</td>
<td>2008H</td>
<td>4</td>
</tr>
<tr>
<td>High Speed Outputs</td>
<td>2007H</td>
<td>2006H</td>
<td>3</td>
</tr>
<tr>
<td>HSI Data Available</td>
<td>2005H</td>
<td>2004H</td>
<td>2</td>
</tr>
<tr>
<td>A/D Conversion Complete</td>
<td>2003H</td>
<td>2002H</td>
<td>1</td>
</tr>
<tr>
<td>Timer Overflow</td>
<td>2001H</td>
<td>2000H</td>
<td>0 (Lowest)</td>
</tr>
</tbody>
</table>

Figure 2-10. Interrupt Vector Locations

![Figure 2-9. Power Down Timing](image)
2.5.2. Interrupt Control

A block diagram of the interrupt system is shown in Figure 2-12. Each of the interrupt sources is tested for a 0 to 1 transition. If this transition occurs, the corresponding bit in the Interrupt Pending Register, located at 0009H, is set. The bit is cleared when the vector is taken to the interrupt routine. Since this register can be written to, it is possible to generate software interrupts by setting bits within the register, or remove pending interrupts by clearing the bits in this register. The pending register can be set even if the interrupt is disabled.

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096 will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096 holds off acknowledging interrupts during these "read/modify/write" instructions.

Enabling and disabling of individual interrupts is done through the Interrupt Mask Register, located at 0008H. If the bit in the mask register is a 1 then the interrupt is enabled, otherwise it is disabled. Even if an interrupt is masked it may still become pending. It may, therefore, be desirable to clear the pending bit before unmasking an interrupt.

The Interrupt Mask Register is also the low byte of the PSW. All of the interrupts may be enabled and disabled simultaneously by using the "EI" (Enable Interrupt) and "DI" (Disable Interrupt) instructions. EI and DI set and clear PSW.9, the interrupt enable bit, they do not effect the contents of the mask register.

2.5.3. Interrupt Priority Programming

The priority encoder looks at all of the interrupts which are both pending and enabled, and selects the one with the highest priority. The priorities are shown in Figure 2-10 (7 is highest, 0 is lowest.) The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

![Diagram of Interrupt Sources](image-url)
At least one instruction in the ISR will always be executed before another interrupt can be acknowledged. Usually this instruction is "DI" or "PUSHF" (push flags). The PUSHF instruction pushes the PSW onto the stack and then clears it.

Clearing the PSW disables all interrupts in two ways; by clearing PSW.9, the interrupt enable bit and because the Interrupt Mask Register is located in bits 0 through 7 of the PSW. The interrupts which should be permitted to interrupt this ISR can then be set in the mask register and an "EI" instruction executed.

By selectively determining which interrupts are enabled or disabled within which interrupt service routines, it is possible to configure the interrupt system in any way one would desire. More information on programming the interrupts can be found under software programming of interrupts, section 3.6.

The last two instructions in an ISR are normally a "POPF" (pop flags), which restores the PSW and, therefore, the interrupt mask register, followed by a 'RET', which restores the Program Counter. Execution will then continue from the point at which the call was forced.
2.5.4. Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

- EI, DI — Enable and Disable Interrupts
- POPF, PUSHF — Pop and Push Flags
- SIGND — Prefix to perform signed multiply and divide (Note that this is not an ASM-96 Mnemonic, but is used for signed multiply and divide)
- TRAP — Software interrupt

When an interrupt is acknowledged, a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 43 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 71 (43 + 4 + 24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled.

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The 'DI', 'PUSHF', 'POPF' and 'TRAP' instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

2.6. TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated "Timer 1", the second, "Timer 2". Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurrences.

2.6.1. Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFFH and should not be used in programs.

2.6.2. Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising and falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOCO.7). To ensure that all CAM entries are checked each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOCO.1, by triggering HSO channel OEH, or by pulling T2RST or HSI.0 high. The HSO and CAM are described in section 2.8. IOCO.3 and IOCO.5 control the resetting of Timer 2. Figure 2-13 shows the different ways of manipulating Timer 2.

2.6.3. Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.5 and IOS1.4, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears the whole byte, including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.
2.6.4. Timer Related Sections
The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The Baud rate generator can use the T2CLK pin as input to its counter. A complete listing of the functions of IOS1, IOC0, and IOC1 are in section 2.13.

2.7. HIGH SPEED INPUTS

The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 share pins with HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 2-14.

2.7.1. HSI Modes
There are 4 possible modes of operation for each of the HSI. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 2-15.

High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time.

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 2-16 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.
2.7.2. HSI FIFO

When an HSI event occurs, a 7x20 FIFO stores the 16 bits of Timer 1 and the 4 bits indicating the state of the 4 HSI lines at the time the status is read. It can take up to 8 state times for this information to reach the holding register. When the FIFO is full, one additional event can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full any additional events will not be recorded.

2.7.3. HSI Interrupts

Interrupts can be generated from the HSI unit in one of two ways, determined by IOC1.7. If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six entries in it. Since all interrupts are rising edge triggered, if IOC1.7 = 1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more. Interrupts can also be generated by pin HSI.0, which has its own interrupt vector.

2.7.4. HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least six entries. If bit 7 is a 1, the FIFO contains at least 1 entry and the holding register has been loaded. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears the entire byte, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value. See Section 3.7.2.

Reading the HSI is done in two steps. First, the HSI Status register is read to obtain the current state of the HSI pins and which pins had changed at the recorded time. The format of the HSI STATUS Register is shown in Figure 2-17. Second, the HSI Time register is read. Reading the Time register unloads one word of the FIFO, so if the Time register is read before the Status register, the information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

If the HSI TIME and Status register are read without the holding register being loaded, the values read will be undeterminate.

It should be noted that many of the Status register conditions are changed by a reset, see section 2.15.2. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13.

2.8. HIGH SPEED OUTPUTS

The High Speed Output unit (HSO) is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, resetting Timer 2, setting 4 software flags, and switching up to 6 output lines. Interrupts can be generated whenever one of these events is triggered. Up to 8 events can be pending at any one time.

2.8.1. HSIO Shared Pins

Two of the 6 output lines (HSO.0 through HSO.5) are shared with the High Speed Input (HSI) lines. HSO.4 and HSO.5 are shared with HSI.2 and HSI.3, respectively. Bits 4 and 6 of the I/O Control Register 1 (IOC1) are used to enable HSO.4 and HSO.5 as outputs.

2.8.2. HSIO CAM

A block diagram of the HSO unit is shown in Figure 2-18. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with a time value every state time. Therefore, it takes 8 state times to compare all CAM registers with a timer.
Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the command to the HSO unit is shown in Figure 2-19.

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available. It can take up to 8 state times for a command to enter the CAM. For this reason, if Timer 1 is being used as the reference, the minimum time that can be loaded is Timer 1 + 1. A similar restriction applies if Timer 2 is used as the reference.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag used in the interrupt routine will be written to the CAM at both the time specified by the interrupt routine and the time specified by the main program. The command tag from the main program will not be executed. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit. See also Section 3.7.3.
2.8.3. HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOS0) bits 6 and 7 indicate the status of the HSO unit. This register is described in section 2.13.4. If IOS0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

One location in the CAM file is checked each state-time. Thus, it takes 8 state-times for the Holding Register to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer changes its value. For this reason, any events scheduled to occur at the same time as a Timer 2 match in the CAM will hold an event pending until a time match occurs, if that match is to a time value on Timer 2 which is never reached, the event will remain pending in the CAM until the part is reset.

Additional caution must be used when Timer 2 is being reset using the HSO unit, since resetting Timer 2 using the HSO is an internal event and can therefore happen at any time within the eight-state-time window. For this reason, any events scheduled to occur at the same time as a Timer 2 reset should be logged into the CAM with a Timer 2 value of zero. When using this method to make a programmable modulo counter, the count will stay at the maximum Timer 2 value only until the Reset T2 command is recognized. The count will stay at zero for the transition which would have changed the count from “N” to zero, and then change to a one on the next transition.

2.8.6. Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such “Software Timers” can be in operation at a time. As each preprogrammed time is reached, the HSO unit sets a Software Timer Flag. If the interrupt bit in the command tag was set then a Software Timer Interrupt will also be generated. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred. See also Section 3.7.4.

If more than one software timer interrupt occurs in the same time frame it is possible that multiple software timer interrupts will be generated.

Each read or test of any bit in IOS1 will clear the whole byte. Be certain to save the byte before testing it unless you are only concerned with 1 bit. See also Section 3.2.2.

A complete listing of the functions of IOS0, IOS1, and IOCl can be found in section 2.13. The Timers are described in section 2.6 and the HSI is described in section 2.7.

2.9. ANALOG INPUTS

The A to D converter on the 8096 provides a 10-bit result on one of 8 input channels. Conversion is done using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on the 8097, 8397, 8095 and 8395 members of the MCS®-96 family.

2.9.1. A/D Accuracy

Each conversion requires 168 state-times (42μS at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held at VCC ± 0.3V with VREF = 5.0 ± 0.5V. The A/D result is calculated from the formula:

\[ 1023 \times \frac{\text{input voltage}-\text{ANGND}}{\text{VREF}-\text{ANGND}} \]

It can be seen from this formula that changes in VREF or ANGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

If high absolute accuracy is needed it may be desirable to use a separate power supply, or power traces, to operate the A/D converter. There is no sample and hold circuit internal to the chip, so the input voltage must be held constant for the entire 168 state times. Examples of connecting the A/D converter to various devices are given in section 4.3.
ARCHITECTURAL OVERVIEW

2.9.2. A/D Commands
Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see section 2.5). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #OFH) triggers it. A to D commands are formatted as shown in Figure 2-20.

The command register is double buffered so it is possible to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

2.9.3. A/D Results
Results of the analog conversions are read from the A/D Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 2-21. Note that the status bit may not be set until 8 state times after the go command. Information on using the HSO is in section 2.8.

2.10. PULSE WIDTH MODULATION OUTPUT (D/A)
Digital to analog conversion can be done with the pulse width modulation output; a block diagram of the circuit is shown in Figure 2-22. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 2-23. Note that when the PWM register equals 00, the output is always low.

The output waveform is a variable duty cycle pulse which repeats every 256 state times (64 μS at 12MHz). Changes
2.11. SERIAL PORT

The serial port is compatible with the MCS-51 serial port. It is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. The serial port registers (SBUF) are both accessed at location 07H. A write to this location accesses the transmit register, and a read accesses a physically separate receive register.

The serial port can operate in 4 modes (explained below). Selection of these modes is done through the Serial Port Status/Control register at location 11H, shown in Figure 2-27.

2.11.1. Serial Port Modes

MODE 0
Mode 0 is a shift register mode. The 8096 outputs a train of 8 shift pulses to an external shift register to clock 8 bits of data into or out of the register from or to the 8096. Serial data enters and exits the 8096 through RXD. TXD outputs the shift clock. 8 bits are transmitted or received, LSB first. A timing diagram of this mode is shown in Figure 2-24. This mode is useful as an I/O expander in which application external shift registers can be used as additional parallel I/O ports. An example of using the port in this mode is given in section 4.5.

Details about the hardware required for smooth, accurate D/A conversion can be found in section 4.3.2. Typically, some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOCl.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOCl is in section 2.13.3.

Figure 2-22. Pulse Width Modulated (D/A) Output

Figure 2-23. Typical PWM Outputs
MODE 1
10-bit frames are transmitted through TXD, and received through RXD: a start bit (0), 8 data bits (LSB first), and a stop bit (1). If PEN = 1 then an even parity bit is transmitted instead of the eighth data bit. This mode is the one commonly used for CRT terminals. The data frame for Mode 1 is shown in Figure 2-25.

MODE 2
11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1 using the TB8 bit. If PEN = 1 the 9th bit will be parity. On receive, the received 9th data bit is stored and the serial port interrupt is activated regardless of its value. The data frame for Modes 2 and 3 is shown in Figure 2-26.

MODE 3
11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1 using the TB8 bit. If PEN = 1 the 9th bit will be parity. On receive, the received 9th data bit is stored and the serial port interrupt is activated regardless of its value. The data frame for Modes 2 and 3 is shown in Figure 2-26.

2.11.2. Multiprocessor Communications
Mode 2 and 3 are provided for multiprocessor communications. In mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in mode 2 will be interrupted by a data frame. An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to mode 3 to receive the coming data frames, while the slaves that were not addressed stay in mode 2 and go on about their business.
2.11.3. Controlling the Serial Port
Control of the Serial Port is done through the Serial Port Control/Status register. The format for the control word is shown in Figure 2-27. Note that reads access only part of the byte, as do writes, and that TB8 is cleared after each byte is transmitted.

In Mode 0, if REN=0, writing to SBUF will start a transmission. Causing a rising edge on REN, or clearing RI with REN=1, will start a reception. Setting REN=0 will stop a reception in progress, and inhibit further receptions. To avoid a partial or complete undesired reception, REN must be set to zero before clearing RI. This can be handled in an interrupt environment by using software flags, or in a straight-line code environment by using the Interrupt Pending register to signal the completion of a receive.

In any mode, it is necessary to set IOCl.5 to a 1 to enable the TXD pin. Some examples of the software involved in using the serial port can be found in section 3.8. More information on IOCl is in section 2.13.3.

2.11.4. Determining Baud Rates
Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times, with a minimum period of 16 XTAL1 cycles.

The unsigned integer represented by the lower 15 bits of the baud rate register defines a number B, where B has a maximum value of 32767. The baud rate for the four serial modes using either XTAL1 or T2CLK as the clock source is given by:

Using XTAL1:
- Mode 0: \( Baud\ Rate = \frac{XTAL1\ frequency}{4 \cdot (B + 1)} \); B ≠ 0
- Others: \( Baud\ Rate = \frac{XTAL1\ frequency}{64 \cdot (B + 1)} \)

Using T2CLK:
- Mode 0: \( Baud\ Rate = \frac{T2CLK\ frequency}{B} \); B ≠ 0
- Others: \( Baud\ Rate = \frac{T2CLK\ frequency}{16 \cdot B} \); B ≠ 0

Note that B cannot equal 0, except when using XTAL1 in other than mode 0.

Common baud rate values, using XTAL1 at 12MHz, are shown below.

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>Baud Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>8137H</td>
<td>8013H</td>
</tr>
<tr>
<td>4800</td>
<td>8270H</td>
<td>8026H</td>
</tr>
<tr>
<td>2400</td>
<td>84E1H</td>
<td>804DH</td>
</tr>
<tr>
<td>1200</td>
<td>89C3H</td>
<td>809BH</td>
</tr>
<tr>
<td>300</td>
<td>A70FH</td>
<td>8270H</td>
</tr>
</tbody>
</table>

The maximum asynchronous baud rate is 187.5 Kbaud, with a 12MHz clock.
2.12. I/O PORTS 0, 1, 2, 3, AND 4

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some output only, some bidirectional and some have alternate functions. Input ports connect to the internal bus through an input buffer. Output ports connect through an output buffer to an internal register that holds the output bits. Bidirectional ports consist of an internal register, an output buffer, and an input buffer.

When an instruction accesses a bidirectional port as a source register, the question often arises as to whether the value that is brought into the CPU comes from the internal port register or from the port pins through the input buffer. In the 8096, the value always comes from the port pins, never from the internal register.

2.12.1. Port 0

Port 0 is an input only port which shares its pins with the analog inputs to the A/D Converter. One can read Port 0 digitally, and/or, by writing the appropriate control bits to the A/D Command Register, select one of the lines of this port to be the input to the A/D Converter. While a conversion is in process, the impedance of the selected line is lower than normal. See the data sheet for the specific values.

2.12.2. Port 1

Port 1 is a quasi-bidirectional I/O port. “Quasi-bidirectional” means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). If the internal pulldown is left off (by writing a 1 to the pin), the pin’s logic level can be controlled by an external pulldown which can either be on (to input a 0) or off (to input a 1). From the user’s point of view the main distinction is that a quasi-bidirectional input will source current while being externally held low and will pull itself high if left alone.

In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time. See also Sections 3.7.1 and 4.2.2.

2.12.3. Port 2

Port 2 is a multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

<table>
<thead>
<tr>
<th>Port</th>
<th>Function</th>
<th>Alternate Function</th>
<th>Controlled by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2.0</td>
<td>output</td>
<td>TXD (serial port transmit)</td>
<td>IOC1.5</td>
</tr>
<tr>
<td>P2.1</td>
<td>input</td>
<td>RXD (serial port receive)</td>
<td>N/A</td>
</tr>
<tr>
<td>P2.2</td>
<td>input</td>
<td>EXTINT (external interrupt)</td>
<td>IOC1.1</td>
</tr>
<tr>
<td>P2.3</td>
<td>input</td>
<td>T2CLK (Timer 2 input)</td>
<td>IOC0.7</td>
</tr>
<tr>
<td>P2.4</td>
<td>input</td>
<td>T2RST (Timer 2 reset)</td>
<td>IOC0.5</td>
</tr>
<tr>
<td>P2.5</td>
<td>output</td>
<td>PWM (pulse-width modulation)</td>
<td>IOC1.0</td>
</tr>
<tr>
<td>P2.6</td>
<td>quasi-bidirectional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.7</td>
<td>quasi-bidirectional</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.12.4. Ports 3 and 4

Ports 3 and 4 have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accessing off-chip memory. If the EA line is low, the pins always act as the System Bus. Otherwise they act as bus pins only during a memory access. If these pins are being used as ports and bus pins, ones must be written to them prior to bus operations.

Strong internal pullups are used during external memory read or write cycles when the pins are used as address or data outputs. At any other time, the internal pullups are...
disabled. The port pins and their system bus functions are shown below:

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>System Bus Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>AD0</td>
</tr>
<tr>
<td>P3.1</td>
<td>AD1</td>
</tr>
<tr>
<td>P3.2</td>
<td>AD2</td>
</tr>
<tr>
<td>P3.3</td>
<td>AD3</td>
</tr>
<tr>
<td>P3.4</td>
<td>AD4</td>
</tr>
<tr>
<td>P3.5</td>
<td>AD5</td>
</tr>
<tr>
<td>P3.6</td>
<td>AD6</td>
</tr>
<tr>
<td>P3.7</td>
<td>AD7</td>
</tr>
<tr>
<td>P4.0</td>
<td>AD8</td>
</tr>
<tr>
<td>P4.1</td>
<td>AD9</td>
</tr>
<tr>
<td>P4.2</td>
<td>AD10</td>
</tr>
<tr>
<td>P4.3</td>
<td>AD11</td>
</tr>
<tr>
<td>P4.4</td>
<td>AD12</td>
</tr>
<tr>
<td>P4.5</td>
<td>AD13</td>
</tr>
<tr>
<td>P4.6</td>
<td>AD14</td>
</tr>
<tr>
<td>P4.7</td>
<td>AD15</td>
</tr>
</tbody>
</table>

2.13. STATUS AND CONTROL REGISTERS

2.13.1. I/O Control Registers
There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

Whenever input lines are switched between two sources, or enabled, it is possible to generate transitions on these lines. This could cause problems with respect to edge sensitive lines such as the HSI lines, Interrupt line, and Timer 2 control lines.

2.13.2. I/O Control Register 0 (IOC0)
IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSO unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 2-28.

2.13.3. I/O Control Register 1 (IOC1)
IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in section 2.5. The positions of the IOC1 control bits are shown in Figure 2-29.

2.13.4. I/O Status Register 0 (IOS0)
There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 2-30.

2.13.5. I/O Status Register 1 (IOS1)
IOS1 is located at 0016H. It contains status bits for the timers and the HSI. Every access of this register clears all of the timer overflow and timer expired bits. It is, therefore, important to first store the byte in a temporary location before attempting to test any bit unless only one bit will ever be of importance to the program. The status bits of IOS1 are shown in Figure 2-31.

2.14. WATCHDOG TIMER (WDT)
This feature is provided as a means of graceful recovery from a software upset. Once the watchdog is initialized, if the software fails to reset the watchdog at least every 64K state times, a hardware reset will be initiated.

The watchdog is initialized by the first write of the clear WDT code to the WDT register. Once the watchdog is initialized it cannot be disabled by software. On reset the watchdog is not active.
The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 ms if a 12 MHz oscillator is used.

The watchdog timer is a 16-bit counter which is incremented every state time. When it overflows it pulls down the RESET pin for at least two state times, resetting the 8096 and any other chips connected to the reset line. To prevent the timer from overflowing and resetting the system, it must be cleared periodically. Clearing the timer is done by writing a "01Eh" followed by an "0E1h" to the WDT register at location 000AH.

Use of a large reset capacitor on the RESET pin will increase the length of time required for a watchdog initiated reset. This is because the capacitor will keep the RESET pin from responding immediately to the internal pull-ups and pull-downs. A large capacitor on the RESET pin may also interfere with the reset of other parts connected to the RESET pin. Under some circumstances, it may be desirable to use an open collector circuit. See section 4.4.

2.14.1. Disabling The Watchdog
The watchdog should be disabled by software not initializing it. If this is not possible, such as during program development, the watchdog can be disabled by holding the RESET pin at 2.0 to 2.5 volts. Voltages over 2.5 volts on the pin could quickly damage the part. Even at 2.5 volts, using this technique for other than debugging purposes is not recommended, as it may affect long term reliability. It is further recommended that any part used in this way for more than several seconds, not be used in production versions of products.

2.15. RESET
2.15.1. Reset Signal
As with all processors, the 8096 must be reset each time the power is turned on. To complete a reset, the RESET pin must be held active (low) for at least 2 state times after VCC, the oscillator, and the back bias generator have stabilized (~1.0 milliseconds). Then when RESET is brought high again, the 8096 executes a reset sequence that takes 10 state times. (It initializes some registers, clears the PSW and jumps to address 2080H.)

The 8096 can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2
state times longer than required for VCC and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire anded together. Remember, the RESET pin itself can be a reset source (see section 2.14). Details of hardware suggestions for reset can be found in section 4.4.

2.15.2. Reset Status

The I/O lines and control of the 8096 will be in their reset state within 2 state times after reset is low, with VCC and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

<table>
<thead>
<tr>
<th>SFR</th>
<th>reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port 1</td>
<td>11111111B</td>
</tr>
<tr>
<td>Port 2</td>
<td>110XXXXX1B</td>
</tr>
<tr>
<td>Port 3</td>
<td>11111111B</td>
</tr>
<tr>
<td>Port 4</td>
<td>11111111B</td>
</tr>
<tr>
<td>PWM Control</td>
<td>00H</td>
</tr>
<tr>
<td>Serial Port (Transmit)</td>
<td>undefined</td>
</tr>
<tr>
<td>Serial Port (Receive)</td>
<td>undefined</td>
</tr>
<tr>
<td>Baud Rate Register</td>
<td>undefined</td>
</tr>
<tr>
<td>Serial Control/Status</td>
<td>undefined</td>
</tr>
<tr>
<td>A/D Command</td>
<td>undefined</td>
</tr>
<tr>
<td>A/D Result</td>
<td>undefined</td>
</tr>
<tr>
<td>Interrupt Pending</td>
<td>undefined</td>
</tr>
<tr>
<td>Interrupt Mask</td>
<td>00000000B</td>
</tr>
<tr>
<td>Timer 1</td>
<td>0000H</td>
</tr>
<tr>
<td>Timer 2</td>
<td>0000H</td>
</tr>
<tr>
<td>Watchdog Timer</td>
<td>0000H</td>
</tr>
<tr>
<td>HSI Mode</td>
<td>11111111B</td>
</tr>
<tr>
<td>HSI Status</td>
<td>undefined</td>
</tr>
<tr>
<td>IOS0</td>
<td>00000000B</td>
</tr>
<tr>
<td>IOS1</td>
<td>00000000B</td>
</tr>
<tr>
<td>IOC0</td>
<td>X00X0X00B</td>
</tr>
<tr>
<td>IOC1</td>
<td>X00X0XX1B</td>
</tr>
</tbody>
</table>

Other conditions following a reset are:

<table>
<thead>
<tr>
<th>Register</th>
<th>reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSI FIFO</td>
<td>empty</td>
</tr>
<tr>
<td>HSO CAM</td>
<td>empty</td>
</tr>
<tr>
<td>HSO lines</td>
<td>000000B</td>
</tr>
<tr>
<td>PSW</td>
<td>0000H</td>
</tr>
<tr>
<td>Stack Pointer</td>
<td>undefined</td>
</tr>
<tr>
<td>Program Counter</td>
<td>2080H</td>
</tr>
<tr>
<td>RD</td>
<td>high</td>
</tr>
<tr>
<td>WR</td>
<td>high</td>
</tr>
<tr>
<td>ALE</td>
<td>low</td>
</tr>
<tr>
<td>BHE</td>
<td>low</td>
</tr>
<tr>
<td>INST</td>
<td>high</td>
</tr>
</tbody>
</table>

NMI

A low to high transition causes a vector to external memory location 0000H. Reserved for use in Intel Development systems.

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

2.15.3. Reset Sync Mode

The RESET line can be used to start the 8096 at an exact state time to provide for synchronization of test equipment and multiple chip systems. RESET is active low. To synchronize parts, RESET is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in section 4.1.5.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

2.16. PIN DESCRIPTION

VCC

Main supply voltage (5V).

VSS

Digital circuit ground (0V). There are two VSS pins, both must be tied to ground.

VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. See section 2.4.2 and 4.

VREF

Reference voltage and power supply for the analog portion of the A/D converter. Nominally at 5 volts. See section 2.9.1 and 4.

ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS. See section 2.9.1 and 4.

VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01 uf capacitor (and not connected to anything else). The capacitor is not required if the A/D converter is not being used.

XTAL1

Input of the oscillator inverter and input to the internal clock generator. See sections 2.2 and 4.
ARCHITECTURAL OVERVIEW

XTAL2
Output of the oscillator inverter. See section 2.2.

CLKOUT
Output of the internal clock generator. The frequency of
CLKOUT is 1/4 the oscillator frequency. It has a 33% duty
cycle. CLKOUT can drive one TTL input. See section 2.2.

RESET
Reset input to the chip, also output to other circuits. Input
low for at least 2 state times to reset the chip. RESET has
a strong internal pullup. See section 2.15 and 4.1.

TEST
Input low enables a factory test mode. The user should
tie this pin to VCC for normal operation.

NMI
A low to high transition causes a vector to external mem-
ory location 0000H. Reserved for use in Intel Develop-
ment systems.

INST
Output high while the address is valid during an external
read indicates the read is an instruction fetch. See section
2.3.6 and 4.6.

EA
Input for memory select (External Access). EA = 1 causes
memory accesses to locations 2000H through 3FFFH to
be directed to on-chip ROM. EA = 0 causes accesses to
their locations to be directed to off-chip memory. EA has
an internal pullup, so it goes to 0 unless driven to 1.
See section 2.3.3.

ALE
Address Latch Enable output. ALE is activated only dur-
ing external memory accesses. It is used to latch the ad-
dress from the multiplexed address/data bus. See section
2.3.5 and 4.6.

RD
Read signal output to external memory. RD is activated
only during external memory reads. See section 2.3.5 and
4.6.

WR
Write signal output to external memory. WR is activated
only during external memory writes. See section 2.3.6
and 4.6.

BHE
Bus High Enable signal output to external memory. BHE
(0/1) selects/deselects the bank of memory that is con-
ected to the high byte of the data bus. See section 2.3.5
and 4.6.

READY
The READY input is used to lengthen external memory
bus cycles up to the time specified in the data sheet. It
has a weak internal pullup. See section 2.3.6 and 4.6.

HSI
High impedance inputs to HSI Unit. Four HSI pins are
available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them
(HSI.2 and HSI.3) are shared with the HSO Unit. See
section 2.7.

HSO
Outputs from HSO Unit. Six HSO pins are available:
HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5.
Two of them (HSO.4 and HSO.5) are shared with the HSI
Unit. All HSO pins are capable of driving one TTL input.
See section 2.8.

PORT 0/ANALOG CHANNEL
High impedance input-only port. These pins can be used
as digital inputs and/or as analog inputs to the on-chip A/D
converter. See sections 2.9 and 2.12.1.

PORT 1
Quasi-bidirectional I/O port. All pins of P1 are capable of
driving one LS TTL input. See section 2.12.2.

PORT 2
Multi-functional port. Six of its pins are shared with other
functions in the 8096, as shown below.

<table>
<thead>
<tr>
<th>Port</th>
<th>Function</th>
<th>Alternate Function</th>
<th>Reference section</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2.0</td>
<td>output</td>
<td>TXD (serial port transmit)</td>
<td>2.11.3</td>
</tr>
<tr>
<td>P2.1</td>
<td>input</td>
<td>RXD (serial port receive)</td>
<td>2.11.3</td>
</tr>
<tr>
<td>P2.2</td>
<td>input</td>
<td>EXTINT (external interrupt)</td>
<td>2.5</td>
</tr>
<tr>
<td>P2.3</td>
<td>input</td>
<td>T2CLK (Timer 2 input)</td>
<td>2.6.2</td>
</tr>
<tr>
<td>P2.4</td>
<td>input</td>
<td>T2RST (Timer 2 reset)</td>
<td>2.6.2</td>
</tr>
<tr>
<td>P2.5</td>
<td>output</td>
<td>PWM (pulse-width modulator)</td>
<td>2.10</td>
</tr>
<tr>
<td>P2.6</td>
<td>quasi-bidirectional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2.7</td>
<td>quasi-bidirectional</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ARCHITECTURAL OVERVIEW

The multi-functional inputs are high impedance. See section 2.12.3.

PORTS 3 AND 4
8-bit bidirectional I/O ports. These pins are shared the multiplexed address/data bus when accessing external memory, with the Port 3 pins accessing the low byte and Port 4 pins accessing the high byte. They are open drain except when being used as system bus pins. See section 2.3.5.

2.17. PIN LIST
The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0-AD15, which are listed under Port 3 and Port 4.

<table>
<thead>
<tr>
<th>Name</th>
<th>68-Pin</th>
<th>48-Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACH0/P0.0</td>
<td>4</td>
<td>—</td>
</tr>
<tr>
<td>ACH1/P0.1</td>
<td>5</td>
<td>—</td>
</tr>
<tr>
<td>ACH2/P0.2</td>
<td>3</td>
<td>—</td>
</tr>
<tr>
<td>ACH3/P0.3</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>ACH4/P0.4</td>
<td>67</td>
<td>43</td>
</tr>
<tr>
<td>ACH5/P0.5</td>
<td>68</td>
<td>42</td>
</tr>
<tr>
<td>ACH6/P0.6</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>ACH7/P0.7</td>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td>ALE</td>
<td>16</td>
<td>34</td>
</tr>
<tr>
<td>ANGND</td>
<td>66</td>
<td>44</td>
</tr>
<tr>
<td>BHE</td>
<td>37</td>
<td>15</td>
</tr>
<tr>
<td>CLKOUT</td>
<td>13</td>
<td>—</td>
</tr>
<tr>
<td>EA</td>
<td>7</td>
<td>39</td>
</tr>
<tr>
<td>EXTINT/P2.2</td>
<td>63</td>
<td>47</td>
</tr>
<tr>
<td>HSI.0</td>
<td>54</td>
<td>3</td>
</tr>
<tr>
<td>HSI.1</td>
<td>53</td>
<td>4</td>
</tr>
<tr>
<td>HSI.2/HSO.4</td>
<td>52</td>
<td>5</td>
</tr>
<tr>
<td>HSI.3/HSO.5</td>
<td>51</td>
<td>6</td>
</tr>
<tr>
<td>HSO.0</td>
<td>50</td>
<td>7</td>
</tr>
<tr>
<td>HSO.1</td>
<td>49</td>
<td>8</td>
</tr>
<tr>
<td>HSO.2</td>
<td>44</td>
<td>9</td>
</tr>
<tr>
<td>HSO.3</td>
<td>43</td>
<td>10</td>
</tr>
<tr>
<td>HSO.4/HSI.2</td>
<td>52</td>
<td>5</td>
</tr>
<tr>
<td>HSO.5/HSI.3</td>
<td>51</td>
<td>6</td>
</tr>
<tr>
<td>INST</td>
<td>15</td>
<td>—</td>
</tr>
<tr>
<td>NMI</td>
<td>7</td>
<td>—</td>
</tr>
<tr>
<td>PWM/P2.5</td>
<td>39</td>
<td>13</td>
</tr>
<tr>
<td>P0.0/ACH0</td>
<td>4</td>
<td>—</td>
</tr>
<tr>
<td>P0.1/ACH1</td>
<td>5</td>
<td>—</td>
</tr>
<tr>
<td>P0.2/ACH2</td>
<td>3</td>
<td>—</td>
</tr>
<tr>
<td>P0.3/ACH3</td>
<td>6</td>
<td>—</td>
</tr>
<tr>
<td>P0.4/ACH4</td>
<td>67</td>
<td>43</td>
</tr>
<tr>
<td>P0.5/ACH5</td>
<td>68</td>
<td>42</td>
</tr>
<tr>
<td>P0.6/ACH6</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>P0.7/ACH7</td>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td>P1.0</td>
<td>59</td>
<td>—</td>
</tr>
</tbody>
</table>

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, TEST, T2CLK(P2.3), T2RST(P2.4).
CHAPTER 3
MCS®-96 SOFTWARE DESIGN INFORMATION

3.0. INTRODUCTION
This section provides information which will primarily interest those who must write programs to execute in the 8096. Several other sources of information are currently available which will also be of interest:

MCS®-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 122048-001

MCS-96 UTILITIES USER'S GUIDE
Order Number 122049-001

MCS-96 MACRO ASSEMBLER AND UTILITIES POCKET REFERENCE
Order Number 122050-001

Throughout this chapter short segments of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16 bit registers.
AL is the low byte of AX, AH is the high byte.
BL is the low byte of BX
CL is the low byte of CX
DL is the low byte of DX

These are the same as the names for the general data registers used in 8086. It is important to note, however, that in the 8096, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within onboard register file.

3.1. OPERAND TYPES
The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "word" is an eight bit unit of data of any type.

3.1.1. Bytes
BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte.

3.1.3. Short-Integers
SHORT-INTEGERS are 8-bit signed variables which can take on the values between -128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

3.1.4. Integers
INTEGERS are 16-bit signed variables which can take on the values between -32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

3.1.5. Bits
BITs are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

3.1.6. Double-Words
DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with INTEL provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in section 3.5.
3.1.7. Long-Integers
LONG-INTEGERS are 32-bit signed variables which can take on the values between \(-2,147,483,648\) and \(2,147,483,647\). The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte. LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in section 3.5.

3.2. OPERAND ADDRESSING
Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be described as they are seen through the assembly language.
The six basic addressing modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

3.2.1. Register-direct References
The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and a register address must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

<table>
<thead>
<tr>
<th>Examples</th>
<th>Register-direct References</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD AX,BX,CX ; AX := BX + CX</td>
<td></td>
</tr>
<tr>
<td>MUL AX,BX ; AX := AX*BX</td>
<td></td>
</tr>
<tr>
<td>INC BL CL ; CL := CL + 1</td>
<td></td>
</tr>
</tbody>
</table>

3.2.2. Indirect References
The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

<table>
<thead>
<tr>
<th>Examples</th>
<th>Indirect References</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD AX,[AX] ; AX := MEM _ WORD(AX)</td>
<td></td>
</tr>
<tr>
<td>ADDB AL,BL,[CX] ; AL := BL + MEM _ BYTE(CX)</td>
<td></td>
</tr>
<tr>
<td>POP [AX] ; MEM _ WORD(AX) := MEM _ WORD(SP), SP := SP + 2</td>
<td></td>
</tr>
</tbody>
</table>

3.2.3. Indirect with Auto-increment References
This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented after it is used to address the operand. If the instruction operates on BYTES or SHORT-INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect addressable will be incremented by two.

<table>
<thead>
<tr>
<th>Examples</th>
<th>Indirect with Auto-increment References</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD AX,[BX] ; AX := MEM _ WORD(BX); BX := BX + 2</td>
<td></td>
</tr>
<tr>
<td>ADD BL,BL,[CX] ; AL := AL + BL + MEM _ BYTE(CX); CX := CX + 1</td>
<td></td>
</tr>
<tr>
<td>PUSH [AX] ; SP := SP - 2; MEM _ WORD(SP) := MEM _ WORD(AX); AX := AX + 2</td>
<td></td>
</tr>
</tbody>
</table>
3.2.4. Immediate References
This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER opera-

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD AX,#340 ; AX:=AX+340</td>
</tr>
<tr>
<td>PUSH #1234H ; SP:=SP-2; MEM WORD(SP):=1234H</td>
</tr>
<tr>
<td>DIVB AX,#10 ; AL:=AX/10; AH:=AX MOD 10</td>
</tr>
</tbody>
</table>

ands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining oper-

3.2.5. Short-indexed References
In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the oper-

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD AX,12[BX] ; AX:=MEM WORD(BX+12)</td>
</tr>
<tr>
<td>MULB AX,BL,3[CX] ; AX:=BL*MEM BYTE(CX+3)</td>
</tr>
</tbody>
</table>

and which will take part in the calculation. Since the eight bit field is sign-extended the effective address can

be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

3.2.6. Long-indexed References
This addressing mode is like the short-indexed mode except that a 16-bit field is taken from the instruction and added to the WORD variable to form the address of the operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must be register-direct references.

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND AX,BX,TABLE[CX] ; AX:=BX AND MEM WORD(TABLE+CX)</td>
</tr>
<tr>
<td>ST AX,TABLE[BX] ; MEM WORD(TABLE+BX):=AX</td>
</tr>
<tr>
<td>ADDB AL,BL,LOOKUP[CX] ; AL:=BL+MEM BYTE(LOOKUP+CX)</td>
</tr>
</tbody>
</table>

3.2.7. ZERO Register Addressing
The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD AX,1234[0] ; AX:=AX+MEM WORD(1234)</td>
</tr>
<tr>
<td>POP 5678[0] ; MEM WORD(5678):=MEM WORD(SP) ; SP:=SP+2</td>
</tr>
</tbody>
</table>

3.2.8. Stack Pointer Register Addressing
The system stack pointer in the 8096 can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

<table>
<thead>
<tr>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH [SP] ; DUPLICATE TOP OF STACK</td>
</tr>
<tr>
<td>LD AX,2[SP] ; AX:=NEXT TO TOP</td>
</tr>
</tbody>
</table>
3.2.9. Assembly Language Addressing Modes
The 8096 assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

Indexed Addressing. The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

3.3 PROGRAM STATUS WORD
The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in figure 3-1. The information in the PSW can be broken down into two basic categories; interrupt control and condition flags. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF).

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>V</th>
<th>VT</th>
<th>C</th>
<th>I</th>
<th>ST</th>
<th>&lt;Interrupt Mask Reg&gt;</th>
</tr>
</thead>
</table>

Figure 3-1. PSW Register

3.3.1. Interrupt Flags
The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT.Mask — address 8) in the on-board register file. Bit 9 in the PSW is the global interrupt enable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT_PENDING register even if they are locked out. Execution of the corresponding service routines will proceed according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in sections 2.5 and 3.6.

3.3.2. Condition Flags
The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

Z. The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N. The N (Negative) flag is set to indicate that the operation generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows.

V. The V (Overflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type.

VT. The VT (Overflow Trap) flag is set whenever the V flag is set but can only be cleared by an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C. The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then C = 0).

ST. The ST (Sticky bit) set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

MULUB AX, CL, DL ; AX := CL * DL
SHR AX, #4 ; Shift right 4 places
If the C flag is set after the shift it indicates that the bits shifted off the end of operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of this information alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

<table>
<thead>
<tr>
<th>C ST</th>
<th>Value of the bits shifted off</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Value = 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 &lt; Value &lt; ½ LSB</td>
</tr>
<tr>
<td>1 0</td>
<td>Value = ½ LSB</td>
</tr>
<tr>
<td>1 1</td>
<td>Value &gt; ½ LSB</td>
</tr>
</tbody>
</table>

### Figure 3-2. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

### 3.4 INSTRUCTION SET

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT-INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of 32 by 16 divides and for shift operations. The remaining operations on 32 bit variables can be implemented by combinations of 16 bit operations. As an example the sequence:

```
ADD    AX,CX
ADDC   BX,DX
```

performs a 32 bit addition, and the sequence

```
SUB    AX,CX
SUBC   BX,DX
```

performs a 32 bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 8096 (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 8096 NORM instruction which normalizes a 32-bit variable and by the existence of the ST flag in the PSW.

In addition to the operations on the various data types, the 8096 supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDs can be converted to DOUBLE-Words by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEgERS can be converted to LONGs with the EXT (sign extend) instruction.

The MCS-96 instructions for addition, subtraction, and comparison do not distinguish between unsigned words and signed integers. Conditional jumps are provided to allow the user to treat the results of these operations as either signed or unsigned quantities. As an example, the CMPB (compare byte) instruction is used to compare both signed and unsigned eight bit quantities. A JH (jump if higher) could be used following the compare if signed operands were involved or a JGT (jump if greater-than) if signed operands were involved.

Tables 3-1 and 3-2 summarize the operation of each of the instructions and Tables 3-3 and 3-4 give the opcode, byte count, and timing information for each of the instructions.
### Table 3-1. Instruction Summary

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Operation (Note 1)</th>
<th>Flags</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD/ADDB</td>
<td>2</td>
<td>D ← D + A</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>ADD/ADDB</td>
<td>3</td>
<td>D ← B + A</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>ADDC/ADDCB</td>
<td>2</td>
<td>D ← D + A + C</td>
<td>✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>SUB/SUBB</td>
<td>2</td>
<td>D ← D - A</td>
<td>✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>SUB/SUBB</td>
<td>3</td>
<td>D ← B - A</td>
<td>✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>SUBC/SUBCB</td>
<td>2</td>
<td>D ← D - A + C - 1</td>
<td>✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>CMP/CMPB</td>
<td>2</td>
<td>D - A</td>
<td>✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>MUL/MULU</td>
<td>2</td>
<td>D, D + 2 ← D * A</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>MUL/MULU</td>
<td>3</td>
<td>D, D + 2 ← B * A</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>MULB/MULUB</td>
<td>2</td>
<td>D, D + 1 ← D * A</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>MULB/MULUB</td>
<td>3</td>
<td>D, D + 1 ← B * A</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>DIV/DIVU</td>
<td>2</td>
<td>D ← (D, D + 2)/A</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>DIVB/DIVUB</td>
<td>2</td>
<td>D ← (D, D + 1)/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND/ANDB</td>
<td>2</td>
<td>D ← D and A</td>
<td>✔ ✔ 0 0</td>
<td></td>
</tr>
<tr>
<td>AND/ANDB</td>
<td>3</td>
<td>D ← B and A</td>
<td>✔ ✔ 0 0</td>
<td></td>
</tr>
<tr>
<td>OR/ORB</td>
<td>2</td>
<td>D ← D or A</td>
<td>✔ ✔ 0 0</td>
<td></td>
</tr>
<tr>
<td>XOR/XORB</td>
<td>2</td>
<td>D ← D (excl. or) A</td>
<td>✔ ✔ 0 0</td>
<td></td>
</tr>
<tr>
<td>LD/LDB</td>
<td>2</td>
<td>D ← A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST/STB</td>
<td>2</td>
<td>A ← D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDBSE</td>
<td>2</td>
<td>D ← A; D + 1 ← SIGN(A)</td>
<td></td>
<td>3,4</td>
</tr>
<tr>
<td>LDBZE</td>
<td>2</td>
<td>D ← A; D + 1 ← 0</td>
<td></td>
<td>3,4</td>
</tr>
<tr>
<td>PUSH</td>
<td>1</td>
<td>SP ← SP - 2; (SP) A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>1</td>
<td>A ← (SP); SP ← SP + 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSHF</td>
<td>0</td>
<td>SP ← SP - 2; (SP) ← PSW; PSW ← 0000H</td>
<td>I ← 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>POPF</td>
<td>0</td>
<td>PSW ← (SP); SP ← SP + 2</td>
<td>I ← ✔ ✔ ✔ ✔ ✔ ✔</td>
<td></td>
</tr>
<tr>
<td>SJMP</td>
<td>1</td>
<td>PC ← PC + 11-bit offset</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>LJMP</td>
<td>1</td>
<td>PC ← PC + 16-bit offset</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>BR(indirect)</td>
<td>1</td>
<td>PC ← (A)</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SCALL</td>
<td>1</td>
<td>SP ← SP - 2; (SP) ← PC; PC ← PC + 11-bit offset</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>LCALL</td>
<td>1</td>
<td>SP ← SP - 2; (SP) ← PC; PC ← PC + 16-bit offset</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>RET</td>
<td>0</td>
<td>PC ← (SP); SP ← SP + 2</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>J(conditional)</td>
<td>1</td>
<td>PC ← PC + 8-bit offset</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>JC</td>
<td>1</td>
<td>Jump if C = 1</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>JNC</td>
<td>1</td>
<td>Jump if C = 0</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

**Note**

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.
## Table 3-2. Instruction Summary

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Operation (Note 1)</th>
<th>Flags</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JE</td>
<td>1</td>
<td>Jump if Z = 1</td>
<td>Z, N</td>
<td>5</td>
</tr>
<tr>
<td>JNE</td>
<td>1</td>
<td>Jump if Z = 0</td>
<td>Z, N</td>
<td>5</td>
</tr>
<tr>
<td>JGE</td>
<td>1</td>
<td>Jump if N = 0</td>
<td>Z, C</td>
<td>5</td>
</tr>
<tr>
<td>JLT</td>
<td>1</td>
<td>Jump if N = 1</td>
<td>Z, C</td>
<td>5</td>
</tr>
<tr>
<td>JGT</td>
<td>1</td>
<td>Jump if N = 0 and Z = 0</td>
<td>Z, C</td>
<td>5</td>
</tr>
<tr>
<td>JLE</td>
<td>1</td>
<td>Jump if N = 1 or Z = 1</td>
<td>Z, C</td>
<td>5</td>
</tr>
<tr>
<td>JH</td>
<td>1</td>
<td>Jump if C = 1 and Z = 0</td>
<td>Z, C</td>
<td>5</td>
</tr>
<tr>
<td>JNH</td>
<td>1</td>
<td>Jump if C = 0 or Z = 1</td>
<td>Z, C</td>
<td>5</td>
</tr>
<tr>
<td>JV</td>
<td>1</td>
<td>Jump if V = 1</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>JNV</td>
<td>1</td>
<td>Jump if V = 0</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>JVT</td>
<td>1</td>
<td>Jump if VT = 1; Clear VT</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>JNVT</td>
<td>1</td>
<td>Jump if VT = 0; Clear VT</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>JST</td>
<td>1</td>
<td>Jump if ST = 1</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>JNST</td>
<td>1</td>
<td>Jump if ST = 0</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>JBS</td>
<td>3</td>
<td>Jump if Specified Bit = 1</td>
<td>Z, V</td>
<td>5,6</td>
</tr>
<tr>
<td>JBC</td>
<td>3</td>
<td>Jump if Specified Bit = 0</td>
<td>Z, V</td>
<td>5,6</td>
</tr>
<tr>
<td>DJNZ</td>
<td>1</td>
<td>D ← D - 1; if D ≠ 0 then</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>DEC/DECB</td>
<td>1</td>
<td>D ← D - 1</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>NEGB</td>
<td>1</td>
<td>D ← 0 - D</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>INC/INCB</td>
<td>1</td>
<td>D ← D + 1</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>EXT</td>
<td>1</td>
<td>D ← D; D + 2 ← Sign (D)</td>
<td>Z, V</td>
<td>2</td>
</tr>
<tr>
<td>EXTB</td>
<td>1</td>
<td>D ← D; D + 1 ← Sign (D)</td>
<td>Z, V</td>
<td>3</td>
</tr>
<tr>
<td>NOT/NOTB</td>
<td>1</td>
<td>D ← Logical Not (D)</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>CLR/CLRB</td>
<td>1</td>
<td>D ← 0</td>
<td>Z, V</td>
<td>5</td>
</tr>
<tr>
<td>SHL/SHLB/SHLL</td>
<td>2</td>
<td>C ← msb -- -- -- -- -- -- -- -- lsb ← 0</td>
<td>Z, V</td>
<td>7</td>
</tr>
<tr>
<td>SHR/SHRB/SHRL</td>
<td>2</td>
<td>0 ← msb -- -- -- -- -- -- -- -- lsb → C</td>
<td>Z, V</td>
<td>7</td>
</tr>
<tr>
<td>SHRA/SHRAB/SHRAL</td>
<td>2</td>
<td>msb → msb -- -- -- -- -- -- -- -- lsb → C</td>
<td>Z, V</td>
<td>7</td>
</tr>
<tr>
<td>SETC</td>
<td>0</td>
<td>C ← 1</td>
<td>Z, N</td>
<td>1</td>
</tr>
<tr>
<td>CLRC</td>
<td>0</td>
<td>C ← 0</td>
<td>Z, N</td>
<td>0</td>
</tr>
<tr>
<td>CLRVT</td>
<td>0</td>
<td>VT ← 0</td>
<td>Z, N</td>
<td>0</td>
</tr>
<tr>
<td>RST</td>
<td>0</td>
<td>PC ← 2080H</td>
<td>Z, N</td>
<td>8</td>
</tr>
<tr>
<td>DI</td>
<td>0</td>
<td>Disable All Interrupts (I ← 0)</td>
<td>Z, N</td>
<td></td>
</tr>
<tr>
<td>EI</td>
<td>0</td>
<td>Enable All Interrupts (I ← 1)</td>
<td>Z, N</td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>0</td>
<td>PC ← PC + 1</td>
<td>Z, N</td>
<td></td>
</tr>
<tr>
<td>SKIP</td>
<td>0</td>
<td>PC ← PC + 2</td>
<td>Z, N</td>
<td></td>
</tr>
<tr>
<td>NORML</td>
<td>2</td>
<td>Normalize (See sec 3.13.66)</td>
<td>Z, N</td>
<td>7</td>
</tr>
<tr>
<td>TRAP</td>
<td>0</td>
<td>SP ← SP - 2; (SP) ← PC; PC ← (2010H)</td>
<td>Z, N</td>
<td>9</td>
</tr>
</tbody>
</table>

Note
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
9. The assembler will not accept this mnemonic.
### Table 3-3. Opcode and State Time Listing

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPERANDS</th>
<th>DIRECT</th>
<th>IMMEDIATE</th>
<th>INDIRECT&lt;sup&gt;©&lt;/sup&gt;</th>
<th>INDEXED&lt;sup&gt;©&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OP CODE</td>
<td>BYTES</td>
<td>STATE TIMES</td>
<td>OP CODE</td>
</tr>
<tr>
<td>ADD</td>
<td>2</td>
<td>64 4 3 4</td>
<td>65 4 5</td>
<td>66 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>ADD</td>
<td>3</td>
<td>44 4 5 5</td>
<td>45 5 6</td>
<td>46 4 7/12</td>
<td>4 8/13</td>
</tr>
<tr>
<td>ADDB</td>
<td>2</td>
<td>74 3 4 4</td>
<td>75 3 4</td>
<td>76 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>ADDB</td>
<td>3</td>
<td>54 4 5 5</td>
<td>55 4 5</td>
<td>56 4 7/12</td>
<td>4 8/13</td>
</tr>
<tr>
<td>ADDC</td>
<td>2</td>
<td>A4 3 4 4</td>
<td>A5 4 5</td>
<td>A6 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>ADDCB</td>
<td>2</td>
<td>B4 3 4 4</td>
<td>B5 4 5</td>
<td>B6 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>SUB</td>
<td>2</td>
<td>68 3 4 4</td>
<td>69 4 5</td>
<td>6A 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>SUB</td>
<td>3</td>
<td>48 3 4 4</td>
<td>49 5 6</td>
<td>4A 4 7/12</td>
<td>4 8/13</td>
</tr>
<tr>
<td>SUBB</td>
<td>2</td>
<td>78 3 4 4</td>
<td>79 3 4</td>
<td>7A 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>SUBB</td>
<td>3</td>
<td>58 4 5 5</td>
<td>59 4 5</td>
<td>5A 4 7/12</td>
<td>4 8/13</td>
</tr>
<tr>
<td>SUBC</td>
<td>2</td>
<td>A8 3 4 4</td>
<td>A9 4 5</td>
<td>AA 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>SUBCB</td>
<td>2</td>
<td>B8 3 4 4</td>
<td>B9 3 4</td>
<td>BA 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>CMP</td>
<td>2</td>
<td>88 3 4 4</td>
<td>89 4 5</td>
<td>8A 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>CMPB</td>
<td>2</td>
<td>98 3 4 4</td>
<td>99 3 4</td>
<td>9A 3 6/11</td>
<td>3 7/12</td>
</tr>
<tr>
<td>MULU</td>
<td>2</td>
<td>6C 3 25</td>
<td>6D 4 26</td>
<td>6E 3 27/32</td>
<td>3 28/33</td>
</tr>
<tr>
<td>MULU</td>
<td>3</td>
<td>4C 3 26</td>
<td>4D 5 27</td>
<td>4E 4 28/33</td>
<td>4 29/34</td>
</tr>
<tr>
<td>MULUB</td>
<td>2</td>
<td>7C 3 17</td>
<td>7D 3 17</td>
<td>7E 3 19/24</td>
<td>3 20/25</td>
</tr>
<tr>
<td>MULUB</td>
<td>3</td>
<td>5C 3 18</td>
<td>5D 4 18</td>
<td>5E 4 20/25</td>
<td>4 21/26</td>
</tr>
<tr>
<td>MUL</td>
<td>2</td>
<td>② 4 29</td>
<td>③ 5 30</td>
<td>④ 3 31/36</td>
<td>4 32/37</td>
</tr>
<tr>
<td>MUL</td>
<td>3</td>
<td>② 5 30</td>
<td>③ 6 31</td>
<td>④ 5 32/37</td>
<td>5 33/38</td>
</tr>
<tr>
<td>MULB</td>
<td>2</td>
<td>② 4 21</td>
<td>③ 4 21</td>
<td>④ 5 24/29</td>
<td>4 24/29</td>
</tr>
<tr>
<td>MULB</td>
<td>3</td>
<td>② 5 22</td>
<td>③ 6 31</td>
<td>④ 5 25/30</td>
<td>⑤ 5 25/30</td>
</tr>
<tr>
<td>DIVU</td>
<td>2</td>
<td>8C 3 25</td>
<td>8D 3 26</td>
<td>8E 3 28/32</td>
<td>3 29/33</td>
</tr>
<tr>
<td>DIVU</td>
<td>2</td>
<td>9C 3 17</td>
<td>9D 3 17</td>
<td>9E 3 20/24</td>
<td>3 21/25</td>
</tr>
<tr>
<td>DIV</td>
<td>2</td>
<td>② 4 29</td>
<td>③ 5 30</td>
<td>④ 3 32/36</td>
<td>4 33/37</td>
</tr>
<tr>
<td>DIV</td>
<td>2</td>
<td>② 4 21</td>
<td>③ 4 21</td>
<td>④ 5 24/28</td>
<td>4 25/29</td>
</tr>
</tbody>
</table>

**Notes:**

① Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

② Number of state times shown for internal/external operands.

③ The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
Table 3-3. Continued

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPERANDS</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATE TIMES</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATE TIMES</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATE TIMES</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATE TIMES</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATE TIMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECT</td>
<td>INDIRECT</td>
<td>INDEXED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORMAL</td>
<td>AUTO-INC.</td>
<td>SHORT</td>
<td>LONG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOGICAL INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND 2 60 3 4 61 4 5</td>
</tr>
<tr>
<td>AND 3 40 4 5</td>
</tr>
<tr>
<td>ANDB 2 70 3 4</td>
</tr>
<tr>
<td>ANDB 3 50 4 5</td>
</tr>
<tr>
<td>OR 2 80 3 4</td>
</tr>
<tr>
<td>ORB 2 90 3 4</td>
</tr>
<tr>
<td>XOR 2 84 3 4</td>
</tr>
<tr>
<td>XORB 2 94 3 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA TRANSFER INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD 2 A0 3 4 A1 4 5 A2 3 6/11 3 7/12 A3 4 6/11 5 7/12</td>
</tr>
<tr>
<td>LDB 2 B0 3 4 B1 3 4 B2 3 6/11 3 7/12 B3 4 6/11 5 7/12</td>
</tr>
<tr>
<td>ST 2 C0 3 4</td>
</tr>
<tr>
<td>STB 2 C4 3 4</td>
</tr>
<tr>
<td>LDBSE 2 BC 3 4 BD 3 4 BE 3 6/11 3 7/12 BF 4 6/11 5 7/12</td>
</tr>
<tr>
<td>LDBZE 2 AC 3 4 AD 3 4 AE 3 6/11 3 7/12 AF 4 6/11 5 7/12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STACK OPERATIONS (internal stack)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH 1 C8 2 8 C9 3 8 CA 2 11/15 2 12/16 CB 3 11/15 4 12/16</td>
</tr>
<tr>
<td>POP 1 CC 2 12 CE 2 14/18 2 14/18 CF 3 14/18 4 14/18</td>
</tr>
<tr>
<td>PUSHF 0 F2 1 8</td>
</tr>
<tr>
<td>POPF 0 F3 1 9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STACK OPERATIONS (external stack)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH 1 C8 2 12 C9 3 12 CA 2 15/19 2 16/20 CB 3 15/19 4 16/20</td>
</tr>
<tr>
<td>POP 1 CC 2 14 CE 2 16/20 2 16/20 CF 3 16/20 4 16/20</td>
</tr>
<tr>
<td>PUSHF 0 F2 1 12</td>
</tr>
<tr>
<td>POPF 0 F3 1 13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JUMPS AND CALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNEMONIC</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>LJMP</td>
</tr>
<tr>
<td>SJMP</td>
</tr>
<tr>
<td>BR[ ]</td>
</tr>
<tr>
<td>TRAP®</td>
</tr>
</tbody>
</table>

Notes:
1. Number of state times shown for internal/external operands.
2. The assembler does not accept this mnemonic.
3. The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
4. State times for stack located internal/external.
5. The assembler uses the generic jump mnemonic (BR) to generate this instruction.
### Table 3-4. CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>JC</td>
<td>DB</td>
<td>JE</td>
<td>DF</td>
<td>JGE</td>
<td>D6</td>
<td>JGT</td>
<td>D2</td>
</tr>
<tr>
<td>JNC</td>
<td>D3</td>
<td>JNE</td>
<td>D7</td>
<td>JLT</td>
<td>DE</td>
<td>JLE</td>
<td>DA</td>
</tr>
<tr>
<td>JH</td>
<td>D9</td>
<td>JV</td>
<td>DD</td>
<td>JVT</td>
<td>DC</td>
<td>JST</td>
<td>D8</td>
</tr>
<tr>
<td>JNH</td>
<td>D1</td>
<td>JNV</td>
<td>D5</td>
<td>JNVT</td>
<td>D4</td>
<td>JNST</td>
<td>D0</td>
</tr>
</tbody>
</table>

### JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>MNEMONIC</th>
<th>OP</th>
<th>BYTE</th>
<th>STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>JBC</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>JBC</td>
<td>31</td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>JBC</td>
<td>32</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>JBC</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>JBC</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>JBC</td>
<td>35</td>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>JBS</td>
<td>38</td>
<td>39</td>
<td>3A</td>
</tr>
<tr>
<td>7</td>
<td>JBS</td>
<td>39</td>
<td>3A</td>
<td>3B</td>
</tr>
<tr>
<td>8</td>
<td>JBS</td>
<td>3A</td>
<td>3B</td>
<td>3C</td>
</tr>
<tr>
<td>9</td>
<td>JBS</td>
<td>3B</td>
<td>3C</td>
<td>3D</td>
</tr>
<tr>
<td>10</td>
<td>JBS</td>
<td>3C</td>
<td>3D</td>
<td>3E</td>
</tr>
<tr>
<td>11</td>
<td>JBS</td>
<td>3D</td>
<td>3E</td>
<td>3F</td>
</tr>
</tbody>
</table>

### LOOP CONTROL

DJNZ  OPCODE EO; 3 BYTES; 5/9 STATE TIMES (NOT TAKEN/TAKEN)

### SINGLE REGISTER INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>05</td>
<td>2</td>
<td>4</td>
<td>EXT</td>
<td>06</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>DECB</td>
<td>15</td>
<td>2</td>
<td>4</td>
<td>EXTB</td>
<td>16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>NEG</td>
<td>03</td>
<td>2</td>
<td>4</td>
<td>NOT</td>
<td>02</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>NEGB</td>
<td>13</td>
<td>2</td>
<td>4</td>
<td>NOTB</td>
<td>12</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>INC</td>
<td>07</td>
<td>2</td>
<td>4</td>
<td>CLR</td>
<td>01</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>INCB</td>
<td>17</td>
<td>2</td>
<td>4</td>
<td>CLR</td>
<td>11</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

### SHIFT INSTRUCTIONS

<table>
<thead>
<tr>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>BYTE</th>
<th>INSTR</th>
<th>DBL</th>
<th>WD</th>
<th>STATE TIMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHL</td>
<td>09</td>
<td>SHLB</td>
<td>19</td>
<td>SHLL</td>
<td>0D</td>
<td>3</td>
<td>7 + 1 PER SHIFT()</td>
</tr>
<tr>
<td>SHR</td>
<td>08</td>
<td>SHRAB</td>
<td>1A</td>
<td>SHRAL</td>
<td>0E</td>
<td>3</td>
<td>7 + 1 PER SHIFT()</td>
</tr>
</tbody>
</table>

### SPECIAL CONTROL INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETC</td>
<td>F9</td>
<td>1</td>
<td>4</td>
<td>DI</td>
<td>FA</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>CLR</td>
<td>F8</td>
<td>1</td>
<td>4</td>
<td>EI</td>
<td>FB</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>CLRVT</td>
<td>FC</td>
<td>1</td>
<td>4</td>
<td>NOP</td>
<td>FD</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>RST</td>
<td>FF</td>
<td>1</td>
<td>16</td>
<td>SKIP</td>
<td>00</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

### NORMALIZE

<table>
<thead>
<tr>
<th>NORMALIZE</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F</td>
<td>3</td>
<td>11 + 1 PER SHIFT</td>
<td></td>
</tr>
</tbody>
</table>
3.5. SOFTWARE STANDARDS AND CONVENTIONS

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programming language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

3.5.1. Register Utilization

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

3.5.2. Addressing 32-bit Operands

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

3.5.3. Subroutine Linkage

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16 bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```plaintext
example_procedure: PROCEDURE (param1,param2,param3);
DECLARE param1 BYTE,
param2 DWORD,
param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

```
?????: param1
high word of param2
low word of param2
param3
return address ← Stack_pointer
```

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8, 16 or 32 bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

a). Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.

b). Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.

c). The Program Status Word (PSW-see section 3.3) is not saved and restored by procedures so the calling code must assume that the condition flags (Z,N,V,VT,C, and ST) are modified by the procedure.

d). Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

3.6. USING THE INTERRUPT SYSTEM

Processing interrupts is an integral part of almost any control application. The 8096 allows the program to manage interrupt servicing in an efficient and flexible manner. Software running in the 8096 exerts control over the interrupt hardware at several levels.
3.6.1. Global Lockout

The processing of interrupts can be enabled or disabled by setting or clearing the I bit in the PSW. This is accomplished by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts; interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

3.6.2. Pending Interrupt Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT_PENDING-register 09H). This register, which has the same bit layout as the interrupt mask register (see next section), can be read or modified as a byte register. This register can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT_PENDING register should ensure that the entire operation is indivisible. The easiest way of doing this is to use the logical instructions in the two or three operand format, as examples:

```
ANDB INT_PENDING,#1111110B ; Clears the A/D interrupt
ORB INT_PENDING,#00000010B ; Sets the A/D interrupt
```

If the required modification to INT_PENDING cannot be accomplished with one instruction then a critical region should be established and the INT_PENDING register modified from within this region (see section 3.6.5).

3.6.3. Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT_MASK-register 08H). The format of this register is shown in figure 3-4.

```
7 6 5 4 3 2 1 0
```

- **TIMER OVERFLOW**
- **A/D COMPLETION**
- **HSI DATA AVAILABLE**
- **HSO EVENT**
- **HSI BIT 0**
- **SOFTWARE TIMERS**
- **SERIAL I/O**
- **EXTERNAL INTERRUPT**

**Figure 3-4. Interrupt Mask Register**

The INT_MASK register can be read or written as a byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The individual masks act like the global lockout in that they only control the servicing of the interrupt; the hardware will save any interrupts that occur in the pending register even if the interrupt mask bit is cleared. The INT_MASK register also can be accessed as the lower eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT_MASK register as well as the global interrupt lockout and the arithmetic flags.

3.6.4. Interrupt Vectors

The 8096 has eight sources of hardware interrupt, each with its own priority and interrupt vector location. Table 3-5 shows the interrupt sources, their priority, and their vector locations. See section 2.5 for a discussion of the various interrupt sources.

```
Table 3-5. Interrupt Vector Information

<table>
<thead>
<tr>
<th>Source</th>
<th>Priority</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer Overflow</td>
<td>0-Lowest</td>
<td>2000H</td>
</tr>
<tr>
<td>A/D Completion</td>
<td>1</td>
<td>2002H</td>
</tr>
<tr>
<td>HSI Data Available</td>
<td>2</td>
<td>2004H</td>
</tr>
<tr>
<td>HSO Execution</td>
<td>3</td>
<td>2006H</td>
</tr>
<tr>
<td>HSI.O</td>
<td>4</td>
<td>2008H</td>
</tr>
<tr>
<td>Software timers</td>
<td>5</td>
<td>200AH</td>
</tr>
<tr>
<td>Serial I/O</td>
<td>6</td>
<td>200CH</td>
</tr>
<tr>
<td>External Interrupt</td>
<td>7-Highest</td>
<td>200EH</td>
</tr>
</tbody>
</table>
```

The programmer must initialize the interrupt vector table with the starting addresses of the appropriate interrupt service routine. It would be a good idea to vector any interrupts that are not used in the system to an error handling routine.

The priorities given in the table give the hardware enforced priorities for these interrupts. This priority controls the order in which pending interrupts are passed to the software via interrupt-calls. The software can implement its own priority structure by controlling the mask register (INT_MASK-register 08H). To see how this is done consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
    PUSHF ; Save the PSW
    (Includes INT_MASK)
    LDB INT_MASK,#0000010B ; Enable interrupts again
    EI
    ; Service the interrupt
    Ret
    ; Restore the PSW
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial_io_isr and the interrupt be enabled for this routine to execute.
There is an interesting chain of instruction side-effects which makes this (or any other) 8096 interrupt service routine execute properly:

a). After the hardware decides to process an interrupt it generates and executes a special interrupt-call instruction which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts the first instruction of the interrupt service routine will execute.

b). The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT_MASK register and the global enable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and by the time the LD instruction starts all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT_MASK instruction.

c). The LD INT_MASK instruction enables those interrupts that the programmer chooses to allow in an interrupt service routine. This example only the INT Dịch vụ sau khi kết thúc sẽ xảy ra các cải biến dữ liệu. Như vậy, cách tốt nhất để giải quyết vấn đề này là sử dụng một biến phức hợp với các biến có thể thay đổi. Việc này sẽ giúp đảm bảo rằng các biến được thay đổi độc lập và không gây ra sự xung đột.

d). The EI instruction reenables the processing of interrupts.

e). The actual interrupt service routine executes within the priority structure established by the software.

f). At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenable an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency. The POPF instruction also pops the interrupt MASK register (part of the PSW), so any changes made to this register during a routine which ends with a POPF will be lost.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

3.6.5. Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

\begin{verbatim}
LDB AL,INT_PENDING
ANDB AL,#bit_mask
STB AL,INT_PENDING
\end{verbatim}

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT_PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT_PENDING contains 00001111B and after the LDB instruction so does AL. IF the HSI interrupt service routine executes at this point then INT_PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT_PENDING to 00000111B. It should be 00000011B. This code sequence has managed to generate a false HSI interrupt! The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction:

\begin{verbatim}
ANDB INT_PENDING,#bit_mask
\end{verbatim}

Instructions are indivisible so mutual exclusion is ensured in this case. For more complex situations, such a simple solution is not available and the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears
the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

3.7. I/O PROGRAMMING CONSIDERATIONS

The on-board I/O devices are, for the most part, simple to program. There are some areas of potential confusion which need to be addressed:

3.7.1. Programming the I/O Ports

Some of the on-board I/O ports can be used as both input and output pins (e.g. Port 1). When the processor writes to the pins of these ports it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to that pin, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with a relatively high impedance pull-up device which can be easily driven down by the device driving the input. If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port. Consider using P1.0 as an input and then trying to toggle P1.1 as an output:

\[
\begin{align*}
\text{ORB} &\text{ IPORT1,} \#00000001B \quad \text{Set P1.0 for input} \\
\text{XORB} &\text{ IPORT1,} \#00000010B \quad \text{Complement P1.1}
\end{align*}
\]

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven high by the 8096 it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7 volts. The 8096 will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin and it will not toggle. The second problem, which is related to the first one, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction then the XORB will write a zero to P1.0 and it will no longer be usable as an input. The first problem can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works. The second problem can be solved in the software fairly easily:

\[
\begin{align*}
\text{LDB} &\text{ AL,} \text{IPORT1} \\
\text{XORB} &\text{ AL,} \#001B \\
\text{ORB} &\text{ AL,} \#001B \\
\text{STB} &\text{ AL,} \text{IPORT1}
\end{align*}
\]

A software solution to both problems is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and then copy it to the port.

3.7.2. Reading the I/O Status Register 1

This status register contains a collection of status flags which relate to the timer and high speed I/O functions (see section 2.12.5). It can be accessed as register 16H in the on-board register file. The layout of this register is shown in figure 3-5. Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

\[
\begin{align*}
\text{LDB} &\text{ AL,IOS1}
\end{align*}
\]

but also to implicit reads such as:

\[
\begin{align*}
\text{JB} &\text{ IOS1.3,} \text{somewhere _ else}
\end{align*}
\]

which jumps to somewhere _ else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which is used to maintain an image of lower five bits of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

\[
\begin{align*}
\text{ORB} &\text{ IOS1 _ image,IOS1}
\end{align*}
\]

leaving IOS1 _ image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1 _ image. Note that if these routines need to clear the flags that they have acted on then the modification of IOS1 _ image must be done from inside a critical region (see section 3.6.5).

3.7.3. Sending Commands to the HSO Unit

Commands are sent to the HSO unit via a byte and then a word write operation:

\[
\begin{align*}
\text{LDB} &\text{ HSO _ COMMAND,} \#\text{what _ to _ do} \\
\text{ADD} &\text{ HSO _ TIME,TIMER1,} \#\text{when _ to _ do _ it}
\end{align*}
\]
The command is actually accepted when the HSO.TIME register is written. It is important to ensure that this code piece is not interrupted by any interrupt service routine which might also send a command to the HSO unit. If this happens the HSO will know when to do it but not know what to do when it’s time to do it. In many systems this becomes a null problem because HSO commands are only issued from one place in the code. If this is not the case then a critical region must be established and the two instructions executed from within this region (see section 3.6.5).

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Flags are available in IOSO which indicate the holding register is empty (IOSO.7) or that both the holding register is empty and the CAM is not full (IOSO.6). The programmer should carefully decide which of these two flags is the best to use for each application.

It is possible to enter commands into the CAM which never execute. This occurs if TIMER2 has been set up as a variable modulo counter and a command is entered with a time tag referenced to TIMER2 which has a value that TIMER2 never reaches. The inaccessible command will never execute and continue to take up room in the CAM until either the system is reset or the program allows TIMER2 increment up to the value stored in the time tag. Note that commands cannot be flushed from the CAM without being executed but that they can be cancelled. This is accomplished by setting the opposite command in the CAM to execute at the same time tag as the command to be cancelled. Since internal events are not synchronized to Timer 1, it is not possible to cancel them. If, as an example, a command has been issued to set HSO.1 when TIMER1 = 1234 then entering a second command which clears HSO.1 when TIMER1 = 1234 will result in a no-operation on HSO.1. Both commands will remain in the CAM until TIMER1 = 1234.

### 3.7.4. High Speed I/O Interrupts

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate on one of the six HSO pins. The other HSO interrupt is the Software Timer interrupt (vector = (200AH)) which is generated (if enabled) for any other HSO command (e.g. triggering the A/D, resetting Timer2 or generating a software time delay).

There are also two interrupts associated with the HSI unit. The HSI data available interrupt (vector = (2004H)) is generated if there is data in the HSI FIFO that the program should read. The other HSI related interrupt is the HSI.0 interrupt which occurs whenever High Speed Input pin 0 makes a zero-to-one transition. This interrupt will become pending in the INT_PENDING register even if the HSI unit is programmed to ignore changes on HSI.0 or look for a one-to-zero transition.

### 3.7.5. Accessing Register Mapped I/O

The on-board I/O devices such as the serial port or the A/D converter are controlled as register mapped I/O. This allows convenient and efficient I/O processing. The implementation of the current members of the MCS-96 family place some restrictions on how these registers can be accessed. While these restrictions are not severe, the programmer must be aware of them. A complete listing of these registers is shown in figure 2-7 and 2-8. The restrictions are as follows:

a). TIMER1, TIMER2 and HSI.TIME are *word read only*. They cannot be read as bytes or written to in any format.

b). HSO.TIME is *word write only*. It cannot be written to as individual bytes or read in any format.

c). RO (the ZERO register) is byte or word read or write but writing to it will not change its value.

d). All of the other I/O registers can be accessed only as bytes. This applies even to the AD_RESULT which is logically a word operand.
### 3.8. EXAMPLE-1 PROGRAMMING THE SERIAL I/O CHANNEL

**MCS-96 MACRO ASSEMBLER SERIAL PORT DEMO PROGRAM**

**SERIES-III MCS-96 MACRO ASSEMBLER, V1.0**

**SOURCE FILE:** :Fl:SPX.SRC  
**OBJECT FILE:** :Fl:SPX.OBJ  
**CONTROLS SPECIFIED IN INVOCATION COMMAND:** NOSB DEBUG

<table>
<thead>
<tr>
<th>ERR LOC</th>
<th>OBJECT</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>rseg</td>
<td>18</td>
<td>$TITLE('SERIAL PORT DEMO PROGRAM')</td>
</tr>
<tr>
<td>0000</td>
<td></td>
<td>19</td>
<td>$PAGELENGTH (95)</td>
</tr>
<tr>
<td>0001</td>
<td></td>
<td>20</td>
<td>This program initializes the serial port and echoes any character sent to it.</td>
</tr>
<tr>
<td>0002</td>
<td></td>
<td>21</td>
<td>; Baud rate = input frequency / (64*baud_val)</td>
</tr>
<tr>
<td>0003</td>
<td></td>
<td>22</td>
<td>; baud_val = (input frequency/64) / baud rate</td>
</tr>
<tr>
<td>0004</td>
<td></td>
<td>23</td>
<td>baud_val equ 39 2400 baud at 6.0 MHz</td>
</tr>
<tr>
<td>0005</td>
<td></td>
<td>24</td>
<td>BAUD := REG, #B000000B</td>
</tr>
<tr>
<td>0006</td>
<td></td>
<td>25</td>
<td>; Set P2.0 to TXD</td>
</tr>
<tr>
<td>0007</td>
<td></td>
<td>26</td>
<td>SBUF, CHR, #00100000B</td>
</tr>
<tr>
<td>0008</td>
<td></td>
<td>27</td>
<td>; Clear serial port</td>
</tr>
<tr>
<td>0009</td>
<td></td>
<td>28</td>
<td>; Continue</td>
</tr>
<tr>
<td>0010</td>
<td></td>
<td>29</td>
<td>The serial port is now initialized</td>
</tr>
<tr>
<td>0011</td>
<td></td>
<td>30</td>
<td>BAUD := REG, #BAUD_LOW</td>
</tr>
<tr>
<td>0012</td>
<td></td>
<td>31</td>
<td>; Enable receiver, Mode 1</td>
</tr>
<tr>
<td>0013</td>
<td></td>
<td>32</td>
<td>; This is necessary because reading SPCON clears TI and RI</td>
</tr>
<tr>
<td>0014</td>
<td></td>
<td>33</td>
<td>; Store byte</td>
</tr>
<tr>
<td>0015</td>
<td></td>
<td>34</td>
<td>; If RI-temp is not set</td>
</tr>
<tr>
<td>0016</td>
<td></td>
<td>35</td>
<td>; Send byte</td>
</tr>
<tr>
<td>0017</td>
<td></td>
<td>36</td>
<td>; Clear bit-received flag</td>
</tr>
</tbody>
</table>

**ASSEMBLY COMPLETED. NO ERROR(S) FOUND.**
3.9. EXAMPLE-2 GENERATING A PWM WITH THE HSO UNIT

MCS-96 MACRO ASSEMBLER HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: f1:HSO2X.SRC
OBJECT FILE: f1:HSO2X.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: HOSB DEBUG

ERR LOC OBJECT LINE SOURCE STATEMENT

3.9. EXAMPLE-2 Generating a PWM with the HSO Unit

The input parameters passed to the program are:

- HSO:TIME
- TIMER1
- HSO:OFF_N
- HSO:ON-1
- HSO:ON-2
- I/O pins

The routine calculates the time for each pin to be high or low based on the input parameters.

Program for PWM Outputs

The input parameters passed to the program are:

- HSO:TIME
- TIMER1
- HSO:OFF_N
- HSO:ON-1
- HSO:ON-2
- I/O pins

The routine calculates the time for each pin to be high or low based on the input parameters.

```
0000 EB00FD E 40 wait: JBS I0SS, 6, wait ; Loop until HSO holding register
0003 FD E 41 NOP
0004 FD E 42 NOP
0005 C701000000 E 43 STB I0SS, D_STAT ; Load byte to external RAM
0006 100100000B E 44 ; For operation with interrupts 'store_stat:' would be the
0007 980100 E 45 ; entry point of the routine.
0008 C7H0100000 E 46 ; Note that a DI or PUSHF might have to be added.
0009 public OLD_STAT
000A 980100 E 47 OLD_STAT: dbs 1 ; Store new status of HSO
000B 980100 E 48 OLD_STAT: ws 1 ; If status hasn't changed
000C 940100 E 49 OLD_STAT: ws 1

PUBLIC wait
0016 300000 R 56 check_0: JBC OLD_STAT, 0, check_1 ; Jump if OLD_STAT(0)=NEW_STAT(0)
0019 380100 R 57 JBS NEW_STAT, 0, set_off_0
001C B10000 E 58 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 0
001F 4701000000000000 E 59 ADD HSO_TIME, TIMER1, HSO OFF_0 ; Time to set pin = Timer1 value
0025 2009 E 60 BR check_1 ; + Time for pin to be low
0027 B10000 E 61 ADD HSO_TIME, TIMER1, HSO OFF_0 ; Set HSO for timer1, clear pin 0
002A 4701000000000000 E 62 ADD HSO_TIME, TIMER1, HSO OFF_0 ; Time to clear pin = Timer1 value
0030 300000 R 63 check_1: JBC OLD_STAT, 1, check_2 ; Jump if OLD_STAT(1)=NEW_STAT(1)
0033 390100 R 64 JBS NEW_STAT, 1, set_off_1
0036 B10000 E 65 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 1
0039 4701000000000000 E 66 ADD HSO_TIME, TIMER1, HSO OFF_1 ; Time to set pin = Timer1 value
003F 2009 E 67 BR check_2 ; + Time for pin to be low
0041 B10000 E 68 ADD HSO_TIME, TIMER1, HSO OFF_1 ; Set HSO for timer1, clear pin 1
0044 4701000000000000 E 69 ADD HSO_TIME, TIMER1, HSO OFF_1 ; Time to clear pin = Timer1 value
004A 300000 R 70 check_2: JBC OLD_STAT, 2, check_3 ; Jump if OLD_STAT(2)=NEW_STAT(2)
004D 390100 R 71 JBS NEW_STAT, 2, set_off_2
004F B10000 E 72 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 2
0052 4701000000000000 E 73 ADD HSO_TIME, TIMER1, HSO OFF_2 ; Time to set pin = Timer1 value
0057 2009 E 74 BR check_3 ; + Time for pin to be low
0059 B10000 E 75 ADD HSO_TIME, TIMER1, HSO OFF_2 ; Set HSO for timer1, clear pin 2
005C 4701000000000000 E 76 ADD HSO_TIME, TIMER1, HSO OFF_2 ; Time to clear pin = Timer1 value
0062 300000 R 77 check_3: JBC OLD_STAT, 3, check_4 ; Jump if OLD_STAT(3)=NEW_STAT(3)
0065 390100 R 78 JBS NEW_STAT, 3, set_off_3
0068 B10000 E 79 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 3
006B 4701000000000000 E 80 ADD HSO_TIME, TIMER1, HSO OFF_3 ; Time to set pin = Timer1 value
006F 2009 E 81 BR check_4 ; + Time for pin to be low
0073 B10000 E 82 ADD HSO_TIME, TIMER1, HSO OFF_3 ; Set HSO for timer1, clear pin 3
0076 4701000000000000 E 83 ADD HSO_TIME, TIMER1, HSO OFF_3 ; Time to clear pin = Timer1 value
007C 300000 R 84 check_4: JBC OLD_STAT, 4, check_5 ; Jump if OLD_STAT(4)=NEW_STAT(4)
007F 390100 R 85 JBS NEW_STAT, 4, set_off_4
0082 B10000 E 86 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 4
0085 4701000000000000 E 87 ADD HSO_TIME, TIMER1, HSO OFF_4 ; Time to set pin = Timer1 value
008A 2009 E 88 BR check_5 ; + Time for pin to be low
008C B10000 E 89 ADD HSO_TIME, TIMER1, HSO OFF_4 ; Set HSO for timer1, clear pin 4
008F 4701000000000000 E 90 ADD HSO_TIME, TIMER1, HSO OFF_4 ; Time to clear pin = Timer1 value
0095 300000 R 91 check_5: JBC OLD_STAT, 5, check_6 ; Jump if OLD_STAT(5)=NEW_STAT(5)
0098 390100 R 92 JBS NEW_STAT, 5, set_off_5
009B B10000 E 93 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 5
009E 4701000000000000 E 94 ADD HSO_TIME, TIMER1, HSO OFF_5 ; Time to set pin = Timer1 value
00A3 2009 E 95 BR check_6 ; + Time for pin to be low
00A6 B10000 E 96 ADD HSO_TIME, TIMER1, HSO OFF_5 ; Set HSO for timer1, clear pin 5
00A9 4701000000000000 E 97 ADD HSO_TIME, TIMER1, HSO OFF_5 ; Time to clear pin = Timer1 value
00AB 300000 R 98 check_6: JBC OLD_STAT, 6, check_7 ; Jump if OLD_STAT(6)=NEW_STAT(6)
00AD 390100 R 99 JBS NEW_STAT, 6, set_off_6
00B0 B10000 E 100 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 6
00B3 4701000000000000 E 101 ADD HSO_TIME, TIMER1, HSO OFF_6 ; Time to set pin = Timer1 value
00B8 2009 E 102 BR check_7 ; + Time for pin to be low
00BB B10000 E 103 ADD HSO_TIME, TIMER1, HSO OFF_6 ; Set HSO for timer1, clear pin 6
00BF 4701000000000000 E 104 ADD HSO_TIME, TIMER1, HSO OFF_6 ; Time to clear pin = Timer1 value
00C3 300000 R 105 check_7: JBC OLD_STAT, 7, check_8 ; Jump if OLD_STAT(7)=NEW_STAT(7)
00C6 390100 R 106 JBS NEW_STAT, 7, set_off_7
00C9 B10000 E 107 HSO COMMAND, #00110000B ; Set HSO for timer1, set pin 7
00CB 4701000000000000 E 108 ADD HSO_TIME, TIMER1, HSO OFF_7 ; Time to set pin = Timer1 value
00CD 2009 E 109 BR check_8 ; + Time for pin to be low
00D0 B10000 E 110 ADD HSO_TIME, TIMER1, HSO OFF_7 ; Set HSO for timer1, clear pin 7
00D3 4701000000000000 E 111 ADD HSO_TIME, TIMER1, HSO OFF_7 ; Time to clear pin = Timer1 value

PUBLIC wait
```

Note that a DI or PUSHF might have to be added.
MCS®-96 SOFTWARE DESIGN INFORMATION

MCS-96 MACRO ASSEMBLER  HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

ERR  LOC  OBJECT  LINE  SOURCE STATEMENT

004A  004A  320017  R  86  check_2:  OLD_STAT, 2, check_3  ; Jump if OLD_STAT(2)=NEW_STAT(2)
004D  3A010B  R  88  JBC  NEW_STAT, 2, set_off_2
0050  90  set_on_2:
0050  B13200  E  91  LDB  HSO_COMMAND, #00110010B  ; Set HSO for timer1, set pin 2
0053  320017  R  92  ADD  HSO_TIME, TIMER1, HSO_OFF_2  ; Time to set pin = Timer1 value
0059  2009  93  BR  check_3  ; + Time for pin to be low
005B  94  set_off_2:
005B  B11200  E  96  LDB  HSO_COMMAND, #00010010B  ; Set HSO for timer1, clear pin 2
005E  470100000000  E  97  ADD  HSO_TIME, TIMER1, HSO_ON_2  ; Time to clear pin = Timer1 value
005F  98  99
0064  100  check_3:
0064  330017  R  102  JBC  OLD_STAT, 3, check_done  ; Jump if OLD_STAT(3)=NEW_STAT(3)
0067  3B010B  R  103  JBS  NEW_STAT, 3, set_off_3
006A  104
006A  B13200  E  106  LDB  HSO_COMMAND, #00110011B  ; Set HSO for timer1, set pin 3
006D  470100000000  E  107  ADD  HSO_TIME, TIMER1, HSO_OFF_3  ; Time to set pin = Timer1 value
0073  2009  108  BR  check_done  ; + Time for pin to be low
0075  109
0075  B11300  E  110  LDB  HSO_COMMAND, #00010011B  ; Set HSO for timer1, clear pin 3
0078  470100000000  E  112  ADD  HSO_TIME, TIMER1, HSO_ON_3  ; Time to clear pin = Timer1 value
007F  113  114  115
007E  116  check_done:
007E  B00100  R  117  LDB  OLD_STAT, NEW_STAT  ; Store current status and
0081  118  F0  RET  wait for interrupt flag
0082  119  120  121

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.
3.10. EXAMPLE-3 MEASURING PULSES WITH THE HSI UNIT

MCS-96 MACRO ASSEMBLER MEASURING PULSES USING THE HSI UNIT

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: FL: PULSEX.SRC
OBJECT FILE: FL: PULSEX.OBJ

CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC OBJECT LINE SOURCE STATEMENT
1 $TITLE('MEASURING PULSES USING THE HSI UNIT')
2 $PAGELENGTH(95)
3 ; This program measures pulsewidths in TIMER1 cycles
4 ; and returns the values in external RAM.
5
6 18H
7 0018 $ SP equ
8 0003 $ HSI_MODE equ 03H
9 0006 $ HSI_STATUS equ 04H
10 0004 $ HSI_TIME equ 04H
11 000A $ TIMER1 equ 0AH
12 0015 $ IOC0 equ 15H
13 0016 $ IOS1 equ 16H
14
15 16
16 0000 X Seg
17 $ reg
18
19 0000 $ HIGH_TIME: dsw l
20 0002 $ LOW_TIME: dsw l
21 0004 $ PERIOD: dsw l
22 0006 $ HI_EDGE: dsw l
23 0008 $ LO_EDGE: dsw l
24
25 000A $ AX: daw 1
26 000A $ AL equ AX :byte
27 000B $ AH equ (AX+l) :byte
28
29 000C $ BX: daw 1
30 000C $ BL equ BX :byte
31 000D $ BU equ (BX+l) :byte ; Note that '88H is an opcode so it
32 ; can't be used as a label
33
34 35 cseg
36
37 0000 A1C00018 $ LD SP, #0COH
38 0004 B10115 $ LDB IOC0, #00000001H ; Enable HSI 0
39 0007 B10F03 $ LDB HSI_MODE, #00001111B ; HSI 0 look for either edge
40
41 000A 44020004 $ R wait: ADD PERIOD, HIGH_TIME, LOW_TIME
42 000B 3716F9 $ JBC IOSl, 7, wait ; Wait while no pulse is entered
43
44 0011 B0060A $ R 45 $ LDB AL, HSI_STATUS ; Load status; Note that reading
45 ; HSI_TIME clears HSI_STATUS
46
47 0014 A0040C $ R 48 $ LD BX, HSI_TIME ; Load the HSI_TIME
48
49 0017 399A09 $ R 50 $ JBS AL, 1, hsi_hi ; Jump if HSI.0 is high
51
52 001A C0060C $ R 53 $ hsi_lo: ST BX, LO_EDGE
53 001D 48060800 $ R 53 $ SUB HSI_TIME, LO_EDGE, HI_EDGE
54 0021 2787 $ R 55 $ BR wait
56
57 0023 C0660C $ R 57 $ hsi_hi: ST BX, HI_EDGE
58
59 0026 48060602 $ R 59 $ SUB LOW_TIME, HI_EDGE, LO_EDGE
60
61 002A 27DB $ R 60 $ BR wait
61
61 002C $ END

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.
3.11. EXAMPLE-4 SCANNING THE A/D CHANNELS

 SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

 CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

 ERL LOC OBJECT

 0000 dseg
 0000 AX: dsw 1
 0000 AL: dsw AX :byte
 0000 AH: (AX+1) :byte
 0000 BX: dsw 1
 0000 BL: (BX+1) :byte
 0001 cseg
 0007 SNAP: LD $1000H ; Set Stack Pointer
 0007 SP: BX, 00H ; Use the zero register
 0007 next: ORB BL, #1000H ; Start conversion on channel
 0008 LDB AD_COMMAND, BL ; indicated by BL register
 0008 ANDB BL, #0111B
 0010 check: JBS AD_RESULT_HI, 3, check ; Wait while A to D is busy
 0011 BR next

 0014 LDB AL, AD_RESULT_LO ; Load low order result
 0017 LDB AH, AD_RESULT_HI ; Load high order result
 001A ADDB DL, BL, BL ; DL=BL*2
 001E LDBZI DX, DL
 0021 ST AX, RESULT_TABLE[DX] ; Store result indexed by BL*2
 0026 INCB BL
 002B ANDB BL, #03H
 002C BR next

 ASSEMBLY COMPLETED, NO ERROR(S) FOUND.

This program scans A to D lines 0 through 3 and stores the results in RESULT_0.

Note that 'BH' is an opcode so it can't be used as a label.

Stack Pointer

Use the zero register

Set Stack Pointer

Start conversion on channel

Wait for conversion to start

Wait while A to D is busy
This program uses a lookup table to generate 12-bit function values using 8-bit input values. The table is 16 bytes long and 16 bits wide. A linear interpolation is made using the following formula:

\[
\frac{\text{TABLE}\_{\text{HIGH}} - \text{TABLE}\_{\text{LOW}}}{16}\ \text{IN}_D\_{\text{IF}} - \text{TABLE}\_{\text{LOW}}\]

Cross Multiplication is used to solve for \(\text{OUT}_D\_{\text{IF}}\):

\[
\text{RESULT}\_{\text{TABLE}}: \text{dsw} \ 1
\]

\[
\text{TABLE}\_{\text{LOW}}:\text{dsb} \ \text{dsw} \ \text{dsw} \ \text{dsw} \ \text{equ} \ (\text{AX}+1) \ \text{byte}
\]

\[
\text{TABLE}\_{\text{HIGH}}: \text{dsb} \ \text{dsw} \ \text{dsw} \ \text{dsw} \ \text{equ} \ (\text{BX}+1) \ \text{byte}
\]

\[
\text{IN}_D\_{\text{IF}}: \text{dsw} \ 1
\]

\[
\text{TAB}\_{\text{DIF}}: \text{dsw} \ 1
\]

\[
\text{OUT}_D\_{\text{IF}}: \text{dsb} \ 1
\]

\[
\text{RESULT}: \text{dsw} \ 1
\]

Note that 'BH' is an opcode so it can't be used as a label.
MC*96 MACRO ASSEMBLER

TABLE LOOKUP AND INTERPOLATION

ERR LOC   OBJECT   LINE   SOURCE  STATEMENT
  0036 0307    82     JNC lab2
  0038 070E    83     INC OUT    ; Round up if Carry = 1
  003A C3010000E R  84     ST OUT, RESULT
  003F 27C3    86     lab2: BR look
  0041
  0042 000000200034004C 91     table: DCW 0000H, 2000H, 3400H, 4C00H   ; A random non-monotonic
  0044 00500004A00720078 92     DCW 5000H, 6A00H, 7200H, 7800H   ; function
  0052 0078007D0076006D 93     DCW 7800H, 7D00H, 7600H, 6000H
  005A 0050004B00340022 94     DCW 5000H, 4B00H, 3400H, 2200H
  0062 0010    95     DCW 1000H
  0064

ASSEMBLY COMPLETED, NO ERROR(S) FOUND.
3.13. DETAILED INSTRUCTION SET DESCRIPTION

This section gives a description of each instruction recognized by the 8096 sorted alphabetically by the mnemonic used in the assembly language for the 8096. Note that the effect on the program counter (PC) is not always shown in the instruction descriptions. All instructions increment the PC by the number of bytes in the instruction. Several acronyms are used in the instruction set descriptions which are defined here:

**aa.** A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

<table>
<thead>
<tr>
<th>aa</th>
<th>Addressing mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Register direct</td>
</tr>
<tr>
<td>01</td>
<td>Immediate</td>
</tr>
<tr>
<td>10</td>
<td>Indirect</td>
</tr>
<tr>
<td>11</td>
<td>Indexed</td>
</tr>
</tbody>
</table>

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

**breg.** A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D."

**baop.** A byte operand which is addressed by any of the address modes discussed in section 3.2.

**bitno.** A three bit field within an instruction op-code which selects one of the eight bits in a byte.

**wreg.** A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D."

**waop.** A word operand which is addressed by any of the address modes discussed in section 3.2.

**Lreg.** A 32-bit register in the internal register file.

**BEA.** Extra bytes of code required for the address mode selected.

**CEA.** Extra state times (cycles) required for the address mode selected.

**cadd** An address in the program code.

**Flag Settings.** The modification to the flag setting is shown for each instruction. A checkmark (✓) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1 or 0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction. A question mark (?) indicates that the flag will be left in an indeterminant state after the operation.

**Generic Jumps and Calls.** The assembler for the 8096 provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user guide (see section 3.0) should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.
3.13.1. ADD (Two Operands) — ADD WORDS

**Operation:** The sum of the two word operands is stored into the destination (leftmost) operand.

\[ (\text{DEST}) \leftarrow (\text{DEST}) + (\text{SRC}) \]

**Assembly Language Format:**

\[
\text{ADD wreg, waop}
\]

**Object Code Format:**

\[
\begin{array}{c}
\text{Bytes: 2 + BEA} \\
\text{States: 4 + CEA}
\end{array}
\]

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3.13.2. ADD (Three Operands) — ADD WORDS

**Operation:** The sum of the second and third word operands is stored into the destination (leftmost) operand.

\[ (\text{DEST}) \leftarrow (\text{SRC1}) + (\text{SRC2}) \]

**Assembly Language Format:**

\[
\text{ADD Dwreg, Swreg, waop}
\]

**Object Code Format:**

\[
\begin{array}{c}
\text{Bytes: 3 + BEA} \\
\text{States: 5 + CEA}
\end{array}
\]

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.3. ADDB (Two Operands) — ADD BYTES

**Operation:** The sum of the two byte operands is stored into the destination (leftmost) operand.

\[(DEST) \leftarrow (DEST) + (SRC)\]

**Assembly Language Format:**

\[
\begin{array}{ccc}
\text{DST} & \text{SRC} \\
\text{ADD Breg, baop} \\
\end{array}
\]

**Object Code Format:**

\[
[011101aa \ \text{baop} \ \text{breg}] \\
\]

Bytes: 2 + BEA
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3.13.4. ADDB (Three Operands) — ADD BYTES

**Operation:** The sum of the second and third byte operands is stored into the destination (leftmost) operand.

\[(DEST) \leftarrow (SRC1) + (SRC2)\]

**Assembly Language Format:**

\[
\begin{array}{ccc}
\text{DST} & \text{SRC1} & \text{SRC2} \\
\text{ADD BDbreg, Sbreg, baop} \\
\end{array}
\]

**Object Code Format:**

\[
[010101aa \ \text{baop} \ \text{Sbreg} \ \text{Dbreg}] \\
\]

Bytes: 3 + BEA
States: 5 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.5. ADDC — ADD WORDS WITH CARRY

**Operation:** The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{DEST}) + (\text{SRC}) + C\]

**Assembly Language Format:**

\[
\text{DST} \quad \text{SRC} \\
\text{ADDC} \quad \text{wreg}, \quad \text{waop}
\]

**Object Code Format:**

\[
[\text{101001aa}] \quad [\text{waop}] \quad [\text{wreg}]
\]

Bytes: 2+BEA  
States: 4+BEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>(\downarrow)</td>
</tr>
</tbody>
</table>

3.13.6. ADDCB — ADD BYTES WITH CARRY

**Operation:** The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{DEST}) + (\text{SRC}) + C\]

**Assembly Language Format:**

\[
\text{DST} \quad \text{SRC} \\
\text{ADDCB} \quad \text{breg}, \quad \text{baop}
\]

**Object Code Format:**

\[
[\text{101101aa}] \quad [\text{baop}] \quad [\text{breg}]
\]

Bytes: 2+BEA  
States: 4+CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>(\downarrow)</td>
</tr>
</tbody>
</table>
3.13.7. AND (Two Operands) — LOGICAL AND WORDS

**Operation:** The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{DEST}) \text{ AND} (\text{SRC})\]

**Assembly Language Format:**

\[
\begin{array}{ll}
\text{DST} & \text{SRC} \\
\text{AND} & \text{wreg, waop}
\end{array}
\]

**Object Code Format:**

\[
\text{[011000a][waop][wreg]}
\]

Bytes: 2 + BEA
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3.13.8. AND (Three Operands) — LOGICAL AND WORDS

**Operation:** The second and third word operands are ANDed; the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{SRC1}) \text{ AND} (\text{SRC2})\]

**Assembly Language Format:**

\[
\begin{array}{llll}
\text{DST} & \text{SRC1} & \text{SRC2} \\
\text{AND} & \text{Dwreg, Swreg, waop}
\end{array}
\]

**Object Code Format:**

\[
\text{[010000a][waop][Swreg][Dwreg]}
\]

Bytes: 3 + BEA
States: 5 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.9. ANDB (Two Operands) — LOGICAL AND BYTES

**Operation:** The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{DEST}) \text{ AND } (\text{SRC})\]

**Assembly Language Format:**

\[
\begin{array}{c}
\text{DST} \\
\text{ANDB} \\
\text{SRC}
\end{array}
\begin{array}{c}
\text{breg,} \\
\text{baop}
\end{array}
\]

**Object Code Format:**

\[
[011100aa][\text{baop}][\text{breg}]
\]

*Bytes: 2 + BEA*

*States: 4 + CEA*

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>√</td>
</tr>
</tbody>
</table>

3.13.10. ANDB (Three Operands) — LOGICAL AND BYTES

**Operation:** The second and third byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{SRC1}) \text{ AND } (\text{SRC2})\]

**Assembly Language Format:**

\[
\begin{array}{c}
\text{DST} \\
\text{ANDB} \\
\text{SRC1} \\
\text{SRC2}
\end{array}
\begin{array}{c}
\text{Dbreg,} \\
\text{Sbreg,} \\
\text{baop}
\end{array}
\]

**Object Code Format:**

\[
[010100aa][\text{baop}][\text{Sbreg}][\text{Dbreg}]
\]

*Bytes: 3 + BEA*

*States: 5 + CEA*

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>√</td>
</tr>
</tbody>
</table>
### 3.13.11. BR (Indirect) — BRANCH INDIRECT

**Operation:** The execution continues at the address specified in the operand word register.

\[ \text{PC} \leftarrow \text{(DEST)} \]

**Assembly Language Format:** \text{BR wreg}

**Object Code Format:** \[ \begin{array}{c} 11100011 \\ \text{[wreg]} \end{array} \]

- **Bytes:** 2
- **States:** 8

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

### 3.13.12. CLR — CLEAR WORD

**Operation:** The value of the word operand is set to zero.

\( \text{(DEST)} \leftarrow 0 \)

**Assembly Language Format:** \text{CLR wreg}

**Object Code Format:** \[ \begin{array}{c} 00000001 \\ \text{[wreg]} \end{array} \]

- **Bytes:** 2
- **States:** 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
3.13.13. CLRB — CLEAR BYTE

Operation: The value of the byte operand is set to zero.

\[(\text{DEST}) \leftarrow 0\]

Assembly Language Format: \text{CLRB \ breg}  

Object Code Format: \[
[ \begin{array}{c}
00010001 \\
\end{array} ] [ \begin{array}{c}
breg \\
\end{array} ]
\]

Bytes: 2  
States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

3.13.14. CLRC — CLEAR CARRY FLAG

Operation: The value of the carry flag is set to zero.

\[C \leftarrow 0\]

Assembly Language Format: \text{CLRC}

Object Code Format: \[
[ \begin{array}{c}
11111000 \\
\end{array} ]
\]

Bytes: 1  
States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
3.13.15. CLRVT — CLEAR OVERFLOW TRAP

**Operation:** The value of the overflow-trap flag is set to zero.

\[ VT \leftarrow 0 \]

**Assembly Language Format:** CLRVT

**Object Code Format:**

\[
\begin{array}{c}
11111100
\end{array}
\]

- **Bytes:** 1
- **States:** 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

3.13.16. CMP — COMPARE WORDS

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

\[ (DEST) \leftarrow (SRC) \]

**Assembly Language Format:**

\[
\begin{array}{c}
\text{CMP } wreg, \text{ waop}
\end{array}
\]

**Object Code Format:**

\[
\begin{array}{c}
100010aa\[]\text{ waop }\[]\text{ wreg }\]
\end{array}
\]

- **Bytes:** \(2 + \text{BEA}\)
- **States:** \(4 + \text{CEA}\)

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✔</td>
</tr>
</tbody>
</table>
3.13.17. CMPB — COMPARE BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format: 

```
CMPB breg, baop
```

Object Code Format: 

```
[ 100110aa ][ baop ][ breg ]
```

Bytes: 2 + BEA
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3.13.18. DEC — DECREMENT WORD

Operation: The value of the word operand is decremented by one.

(DEST) ← (DEST) — 1

Assembly Language Format: 

```
DEC wreg
```

Object Code Format: 

```
[ 00000101 ][ wreg ]
```

Bytes: 2
States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.19. DECB — DECREMENT BYTE

**Operation:** The value of the byte operand is decremented by one.

\[(\text{DEST}) \leftarrow (\text{DEST}) - 1\]

**Assembly Language Format:** `DECB breg`

**Object Code Format:**
- `00010101`
- `breg`
- Bytes: 2
- States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3.13.20. DI — DISABLE INTERRUPTS

**Operation:** Interrupts are disabled. Interrupt-calls will not occur after this instruction.

Interrupt Enable (PSW.9) \(\leftarrow 0\)

**Assembly Language Format:** `DI`

**Object Code Format:**
- `11111010`
- Bytes: 1
- States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>–</td>
</tr>
</tbody>
</table>
3.13.21. DIV — DIVIDE INTEGERS

Operation: This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

\[(\text{low word } \text{DEST}) \leftarrow (\text{DEST}) / (\text{SRC})\]
\[(\text{high word } \text{DEST}) \leftarrow (\text{DEST}) \mod (\text{SRC})\]

The above two statements are performed concurrently.

Assembly Language Format: \[\text{DST SRC } \text{DIV } \text{Ireg, waop}\]

Object Code Format: \[11111110 \ 100011aa \ 1 \ \text{waop } \ 1 \ \text{Ireq}\]

Bytes: 2 + BEA
States: 29 + CEA

Flags Affected

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

3.13.22. DIVB — DIVIDE SHORT-INTEGERS

Operation: This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

\[(\text{low byte } \text{DEST}) \leftarrow (\text{DEST}) / (\text{SRC})\]
\[(\text{high byte } \text{DEST}) \leftarrow (\text{DEST}) \mod (\text{SRC})\]

The above two statements are performed concurrently.

Assembly Language Format: \[\text{DST SRC } \text{DIVB wreg, baop}\]

Object Code Format: \[11111110 \ 100111aa \ 1 \ \text{baop } \ 1 \ \text{wreg}\]

Bytes: 2 + BEA
States: 21 + CEA

Flags Affected

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
3.13.23. DIVU — DIVIDE WORDS

**Operation:** This instruction divides the contents of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order byte will contain the remainder.

\[(\text{low word } \text{DEST}) \leftarrow (\text{DEST}) \div (\text{SRC})\]
\[(\text{high word } \text{DEST}) \leftarrow \text{MOD} (\text{SRC})\]

The above two statements are performed concurrently.

**Assembly Language Format:**

```
DST SRC
DIVU Ireg, waop
```

**Object Code Format:**

```
[ 100011aa ][ waop ][ Ireg ]
```

- Bytes: 2 + BEA
- States: 25 + CEA

---

### Flags Affected

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>↑</td>
<td>-</td>
</tr>
</tbody>
</table>

3.13.24. DIVUB — DIVIDE BYTES

**Operation:** This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

\[(\text{low byte } \text{DEST}) \leftarrow (\text{DEST}) \div (\text{SRC})\]
\[(\text{high byte } \text{DEST}) \leftarrow (\text{DEST}) \mod (\text{SRC})\]

The above two statements are performed concurrently.

**Assembly Language Format:**

```
DST SRC
DIVUB wreg, baop
```

**Object Code Format:**

```
[ 100111aa ][ baop ][ wreg ]
```

- Bytes: 2 + BEA
- States: 17 + CEA

---

### Flags Affected

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>↑</td>
<td>-</td>
</tr>
</tbody>
</table>
3.13.25. DJNZ — DECREMENT AND JUMP IF NOT ZERO

**Operation:** The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

\[
\text{(COUNT)} \leftarrow \text{(COUNT)} - 1 \\
\text{if } (\text{COUNT}) \neq 0 \text{ then} \\
\quad \text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)} \\
\text{end if}
\]

**Assembly Language Format:** DJNZ breg, cadd

**Object Code Format:** [ 11100000 ][ breg ][ disp ]

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>—</td>
</tr>
</tbody>
</table>

3.13.26. EI — ENABLE INTERRUPTS

**Operation:** Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.9) \(\leftarrow 1\)

**Assembly Language Format:** EI

**Object Code Format:** [ 11110111 ]

Bytes: 1
States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>—</td>
</tr>
</tbody>
</table>
3.13.27. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER

Operation: The low order word of the operand is sign-extended throughout the high order word of the operand.

if (low word DEST)<8000H then
    (high word DEST) ← 0
else
    (high word DEST) ← 0FFFFH
end_if

Assembly Language Format: EXT  Ireg

Object Code Format: [ 00000110 ][ Ireg ]

Bytes: 2
States: 4

---

3.13.28. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER

Operation: The low order byte of the operand is sign-extended throughout the high order byte of the operand.

if (low byte DEST)<80H then
    (high byte DEST) ← 0
else
    (high byte DEST) ← 0FFH
end_if

Assembly Language Format: EXTB  wreg

Object Code Format: [ 00010110 ][ wreg ]

Bytes: 2
States: 4
3.13.29. INC — INCREMENT WORD

**Operation:** The value of the word operand is incremented by 1.

\[(\text{DEST}) \leftarrow (\text{DEST}) + 1\]

**Assembly Language Format:** INC wreg

**Object Code Format:** 

\[
\begin{array}{c}
00000111 \\
\end{array} 
\]

Bytes: 2
States: 4

---

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

3.13.30. INCB — INCREMENT BYTE

**Operation:** The value of the byte operand is incremented by 1.

\[(\text{DEST}) \leftarrow (\text{DEST}) + 1\]

**Assembly Language Format:** INCB breg

**Object Code Format:** 

\[
\begin{array}{c}
00010111 \\
\end{array} 
\]

Bytes: 2
States: 4

---

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

---
3.13.31. JBC — JUMP IF BIT CLEAR

Operation: The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

\[
\text{if (specified bit) = 0 then} \\
\quad \text{PC} \leftarrow \text{PC} + \text{disp (sign-extended to 16 bits)}
\]

Assembly Language Format: JBC breg,bitno,cadd

Object Code Format: [ 00110bbb ][ breg ][ disp ]

where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

3-39
3.13.32. JBS — JUMP IF BIT SET

**Operation:** The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

```plaintext
if (specified bit) = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

**Assembly Language Format:** JBS breg,bitno,cadd

**Object Code Format:**

```
00111111 1 [ breg ] [ disp ]
```

where bbb is the bit number within the specified register.

| Bytes: | 3 |
| States: Jump Not Taken | 5 |
| Jump Taken | 9 |

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>

3.13.33. JC — JUMP IF CARRY FLAG IS SET

**Operation:** If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

```plaintext
if C = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

**Assembly Language Format:** JC cadd

**Object Code Format:**

```
110111011 1 [ disp ]
```

| Bytes: | 2 |
| States: Jump Not Taken | 4 |
| Jump Taken | 8 |

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>
3.13.34. **JE — JUMP IF EQUAL**

**Operation:** If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of \(-128\) to \(+127\). If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

\[
\text{if } Z = 1 \text{ then} \quad \text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

**Assembly Language Format:** 

`JE cadd`

**Object Code Format:**

\[
\begin{array}{c|c}
\text{Bytes:} & 2 \\
\text{States: Jump Not Taken:} & 4 \\
\text{Jump Taken:} & 8 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

3.13.35. **JGE — JUMP IF SIGNED GREATER THAN OR EQUAL**

**Operation:** If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of \(-128\) to \(+127\). If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

\[
\text{if } N = 0 \text{ then} \quad \text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

**Assembly Language Format:** 

`JGE cadd`

**Object Code Format:**

\[
\begin{array}{c|c}
\text{Bytes:} & 2 \\
\text{States: Jump Not Taken:} & 4 \\
\text{Jump Taken:} & 8 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3.13.36. JGT — JUMP IF SIGNED GREATER THAN

**Operation:** If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

If $N = 0$ AND $Z = 0$ then

$$PC \leftarrow PC + \text{disp} \text{ (sign-extended to 16 bits)}$$

**Assembly Language Format:** JGT cadd

**Object Code Format:**

$$[11010010][\text{disp}]$$

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

3.13.37. JH — JUMP IF HIGHER (UNSIGNED)

**Operation:** If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

If $C = 1$ and $Z = 0$ then

$$PC \leftarrow PC + \text{disp} \text{ (sign-extended to 16 bits)}$$

**Assembly Language Format:** JH cadd

**Object Code Format:**

$$[11011001][\text{disp}]$$

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
3.13.38. JLE — JUMP IF SIGNED LESS THAN OR EQUAL

Operation: If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If both the negative flag and the zero flag are clear (i.e., 0), control passes to the next sequential instruction.

\[
\text{if } N = 1 \text{ OR } Z = 1 \text{ then } \\
\text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

Assembly Language Format: \texttt{JLE \ cadd}

Object Code Format: [ \texttt{11011010} ][ \texttt{disp} ]

| Bytes: | 2 |
| States: | Jump Not Taken: 4 |
| Jump Taken: | 8 |

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

3.13.39. JLT — JUMP IF SIGNED LESS THAN

Operation: If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

\[
\text{if } N = 1 \text{ then } \\
\text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

Assembly Language Format: \texttt{JLT \ cadd}

Object Code Format: [ \texttt{11011110} ][ \texttt{disp} ]

| Bytes: | 2 |
| States: | Jump Not Taken: 4 |
| Jump Taken: | 8 |

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3.13.40. JNC — JUMP IF CARRY FLAG IS CLEAR

**Operation:** If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

\[
\text{if } C = 0 \text{ then} \\
\text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

**Assembly Language Format:** JNC cadd

**Object Code Format:** [11010011][disp]

| Bytes: | 2 |
| States: Jump Not Taken: | 4 |
| Jump Taken: | 8 |

**Flags Affected**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

3.13.41. JNE — JUMP IF NOT EQUAL

**Operation:** If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

\[
\text{if } Z = 0 \text{ then} \\
\text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

**Assembly Language Format:** JNE cadd

**Object Code Format:** [11010111][disp]

| Bytes: | 2 |
| States: Jump Not Taken: | 4 |
| Jump Taken: | 8 |

**Flags Affected**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3.13.42. JNH — JUMP IF NOT HIGHER (UNSIGNED)

**Operation:** If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of $-128$ to $+127$. If the carry flag is set (i.e., 1), or the zero flag is not, control passes to the next sequential instruction.

If $C = 0$ OR $Z = 1$ then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JNH  cadd

**Object Code Format:**

```
[ 11010001 ][ disp ]
```

**States:**
- Jump Not Taken: 4
- Jump Taken: 8

**Flags Affected:**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.13.43. JNST — JUMP IF STICKY BIT IS CLEAR

**Operation:** If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of $-128$ to $+127$. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

If $ST = 0$ then

PC ← PC + disp (sign-extended to 16 bits)

**Assembly Language Format:** JNST  cadd

**Object Code Format:**

```
[ 11010000 ][ disp ]
```

**States:**
- Jump Not Taken: 4
- Jump Taken: 8

**Flags Affected:**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.13.44. JNV — JUMP IF OVERFLOW FLAG IS CLEAR

**Operation:** If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

\[
\text{if } V = 0 \text{ then} \\
\quad \text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

**Assembly Language Format:** JNV  cadd

**Object Code Format:**

\[
[\ 11010101\ ][\ \text{disp}\ ]
\]

*Bytes:* 2  
*States:* Jump Not Taken: 4  
*Jump Taken:* 8

---

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

---

3.13.45. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR

**Operation:** If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction.

\[
\text{if } VT = 0 \text{ then} \\
\quad \text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

**Assembly Language Format:** JNVT  cadd

**Object Code Format:**

\[
[\ 11010100\ ][\ \text{disp}\ ]
\]

*Bytes:* 2  
*States:* Jump Not Taken: 4  
*Jumps Taken:* 8

---

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
3.13.46. JST — JUMP IF STICKY BIT IS SET

Operation: If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

\[
\text{if } ST = 1 \text{ then} \\
\quad \text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

Assembly Language Format: \text{JST} \text{ cadd}

Object Code Format: \[
\begin{array}{c}
11011000 \\
\text{disp}
\end{array}
\]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

\[
\begin{array}{cccccc}
Z & N & C & V & VT & ST \\
- & - & - & - & - & -
\end{array}
\]

3.13.47. JV — JUMP IF OVERFLOW FLAG IS SET

Operation: If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

\[
\text{if } V = 1 \text{ then} \\
\quad \text{PC} \leftarrow \text{PC} + \text{disp} \text{ (sign-extended to 16 bits)}
\]

Assembly Language Format: \text{JV} \text{ cadd}

Object Code Format: \[
\begin{array}{c}
11011101 \\
\text{disp}
\end{array}
\]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

\[
\begin{array}{cccccc}
Z & N & C & V & VT & ST \\
- & - & - & - & - & -
\end{array}
\]
3.13.48. JVT — JUMP IF OVERFLOW TRAP IS SET

Operation: If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction.

if VT = 1 then
    PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: JVT cadd

Object Code Format: [11011100][disp]

Bytes: 2
States: Jump Not Taken: 4
        Jump Taken: 8

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

3.13.49. LCALL — LONG CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

SP ← SP - 2
(SP) ← PC
PC ← PC + disp

Assembly Language Format: LCALL cadd

Object Code Format: [11101111][disp-low][disp-hi]

Bytes: 3
States: Onchip stack: 13
        Offchip stack: 16

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.13.50. LD — LOAD WORD

**Operation:** The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{SRC})\]

**Assembly Language Format:**

\[
\text{LD } \text{wreg, } \text{waop}
\]

**Object Code Format:**

\[
\begin{array}{ccccccc}
101000 & \text{waop} & 0 & 0 & 0
\end{array}
\]

Bytes: 2 + BEA  
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>

3.13.51. LDB — LOAD BYTE

**Operation:** The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

\[(\text{DEST}) \leftarrow (\text{SRC})\]

**Assembly Language Format:**

\[
\text{LDB } \text{breg, } \text{baop}
\]

**Object Code Format:**

\[
\begin{array}{ccccccc}
101100 & \text{baop} & 0 & 0 & 0
\end{array}
\]

Bytes: 2 + BEA  
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>
3.13.52. LDBSE — LOAD INTEGER WITH SHORT-INTEGER

**Operation:** The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

\[(\text{low byte DEST}) \leftarrow (\text{SRC})\]

\[\text{if } (\text{SRC}) < 80H \text{ then}\]

\[\text{(high byte DEST)} \leftarrow 0\]

\[\text{else}\]

\[\text{(high byte DEST)} \leftarrow 0\text{FFH}\]

end _if

**Assembly Language Format:**

\[
\begin{align*}
\text{DST} & \quad \text{SRC} \\
\text{LDBSE} & \quad \text{wreg}, \quad \text{baop}
\end{align*}
\]

**Object Code Format:**

\[
\begin{align*}
[ & \quad \text{101111aa} ] [ \quad \text{baop} ] [ \quad \text{wreg} ]
\end{align*}
\]

Bytes: 2 + BEA
States: 4 + CEA

---

### Flags Affected

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

3.13.53. LDBZE — LOAD WORD WITH BYTE

**Operation:** The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

\[(\text{low byte DEST}) \leftarrow (\text{SRC})\]

\[(\text{high byte DEST}) \leftarrow 0\]

**Assembly Language Format:**

\[
\begin{align*}
\text{DST} & \quad \text{SRC} \\
\text{LDBZE} & \quad \text{wreg}, \quad \text{baop}
\end{align*}
\]

**Object Code Format:**

\[
\begin{align*}
[ & \quad \text{101011aa} ] [ \quad \text{baop} ] [ \quad \text{wreg} ]
\end{align*}
\]

Bytes: 2 + BEA
States: 4 + CEA

---

### Flags Affected

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3.13.54. LJMP — LONG JUMP

**Operation:** The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

\[
\text{PC} \leftarrow \text{PC} + \text{disp}
\]

**Assembly Language Format:** LJMP cadd

**Object Code Format:**
\[
[ 11100111 ] [ \text{disp-low} ] [ \text{disp-hi} ]
\]

Bytes: 3
States: 8

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

3.13.55. MUL (Two Operands) — MULTIPLY INTEGERS

**Operation:** The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

\[
(\text{DEST}) \leftarrow (\text{DEST}) \ast (\text{SRC})
\]

**Assembly Language Format:**
\[
\text{DST SRC MUL Ireg, waop}
\]

**Object Code Format:**
\[
[ 11111110 ] [ 011011aa ][ \text{waop} ][ \text{Ireg} ]
\]

Bytes: 3 + BEA
States: 29 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
3.13.56. MUL (Three Operands) — MULTIPLY INTEGERS

Operation: The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:  
DST SRC1 SRC2  
MUL  Ireg, wreg, waop

Object Code Format:  
\[ \text{Bytes: 4 + BEA} \]  
\[ \text{States: 30 + CEA} \]

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>

3.13.57. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS

Operation: The two SHORT-INTEGERS are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:  
DST SRC  
MULB  wreg, baop

Object Code Format:  
\[ \text{Bytes: 3 + BEA} \]  
\[ \text{States: 21 + CEA} \]

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>
3.13.58. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

Operation: The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

\[(\text{DEST}) \leftarrow (\text{SRC1}) \times (\text{SRC2})\]

Assembly Language Format:

\[
\begin{array}{ccc}
\text{DST} & \text{SRC1} & \text{SRC2} \\
\text{MULB} & \text{wreg}, & \text{breg} & \text{baop} \\
\end{array}
\]

Object Code Format:

\[
[11111110 \ 010111aa \ \text{baop} \ \text{breg} \ \text{wreg}]
\]

Bytes: 4 + BEA
States: 22 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

3.13.59. MULU (Two Operands) — MULTIPLY WORDS

Operation: The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

\[(\text{DEST}) \leftarrow (\text{DEST}) \times (\text{SRC})\]

Assembly Language Format:

\[
\begin{array}{ccc}
\text{DST} & \text{SRC} \\
\text{MULU} & \text{Ireg}, & \text{waop} \\
\end{array}
\]

Object Code Format:

\[
[011011aa \ \text{waop} \ \text{Ireg}]
\]

Bytes: 2 + BEA
States: 25 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
3.13.60. MULU (Three Operands) — MULTIPLY WORDS

**Operation:** The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

\[(\text{DEST}) < - (\text{SRC1}) \times (\text{SRC2})\]

**Assembly Language Format:**

```
DST SRC1 SRC2
MULU Ireg, wreg, waop
```

**Object Code Format:**

```
[ 010011aa ][ waop ][ wreg ][ lreg ]
```

Bytes: 3 + BEA
States: 26 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

3.13.61. MULUB (Two Operands) — MULTIPLY BYTES

**Operation:** The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

\[(\text{DEST}) \leftarrow (\text{DEST}) \times (\text{SRC})\]

**Assembly Language Format:**

```
DST SRC
MULUB wreg, baop
```

**Object Code Format:**

```
[ 011111aa ][ baop ][ wreg ]
```

Bytes: 2 + BEA
States: 17 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>
3.13.62. MULUB (Three Operands) — MULTIPLY BYTES

**Operation:** The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

\[(DEST) \leftarrow (SRC1) \times (SRC2)\]

**Assembly Language Format:**

```
DST SRC1 SRC2
MULUB wreg, breg, baop
```

**Object Code Format:**

```
[ 010111 aa ][ baop ][ breg ][ wreg ]
```

Bytes: 3 + BEA
States: 18 + CEA

Flag Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.13.63. NEG — NEGATE INTEGER

**Operation:** The value of the INTEGER operand is negated.

\[(DEST) \leftarrow -(DEST)\]

**Assembly Language Format:**

```
NEG wreg
```

**Object Code Format:**

```
[ 00000011 ][ wreg ]
```

Bytes: 2
States: 4

Flag Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.13.64. **NEGB — NEGATE SHORT-INTEGER**

**Operation:** The value of the SHORT-INTEGER operand is negated.

\[(\text{DEST}) \leftarrow - (\text{DEST})\]

**Assembly Language Format:** NEGB  breg

**Object Code Format:** [ 00010011 ][ breg ]

| Bytes: 2 | States: 4 |

---

**Flags Affected**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

3.13.65. **NOP — NO OPERATION**

**Operation:** Nothing is done. Control passes to the next sequential instruction.

**Assembly Language Format:** NOP

**Object Code Format:** [ 11111101 ]

| Bytes: 1 | States: 4 |

---

**Flags Affected**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.13.66. NORML — NORMALIZE LONG-INTEGER

**Operation:** The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

\[
\text{(COUNT)} \leftarrow 0 \\
\text{do while (MSB(DEST) = 0) AND ((COUNT) < 31)} \\
\quad (\text{DEST}) \leftarrow (\text{DEST}) \times 2 \\
\quad (\text{COUNT}) \leftarrow (\text{COUNT}) + 1 \\
\text{end while}
\]

**Assembly Language Format:** NORML ireg,breg

**Object Code Format:** [ 00001111 ][ breg ][ ireg ]

- Bytes: 3
- States: 8 + No. of shifts performed

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3.13.67. NOT — COMPLEMENT WORD

**Operation:** The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

\[
(\text{DEST}) \leftarrow \text{NOT(DEST)}
\]

**Assembly Language Format:** NOT wreg

**Object Code Format:** [ 00000010 ][ wreg ]

- Bytes: 2
- States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.68. NOTB — COMPLEMENT BYTE

**Operation:** The value of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

\[(\text{DEST}) \leftarrow \text{NOT} (\text{DEST})\]

**Assembly Language Format:** NOTB breg

**Object Code Format:**

```
[ 00010010 ] [ breg ]
```

- **Bytes:** 2
- **States:** 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3.13.69. OR — LOGICAL OR WORDS

**Operation:** The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

\[(\text{DEST}) \leftarrow (\text{DEST}) \text{ OR} (\text{SRC})\]

**Assembly Language Format:**

```
DST SRC  
OR wreg, waop
```

**Object Code Format:**

```
[ 100000aa ] [ waop ] [ wreg ]
```

- **Bytes:** 2 + BEA
- **States:** 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.70. ORB — LOGICAL OR BYTES

**Operation:** The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

**Assembly Language Format:** ORB breg,baop

**Object Code Format:** [ 100100aa ][ baop ][ breg ]

Bytes: 2 + BEA  
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>√</td>
</tr>
</tbody>
</table>

3.13.71. POP — POP WORD

**Operation:** The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)  
SP ← SP + 2

**Assembly Language Format:** POP waop

**Object Code Format:** [ 110011aa ][ waop ]

Bytes: 1 + BEA  
States: Onchip Stack: 12 + CEA  
Offchip Stack 14 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
3.13.72. POPF — POP FLAGS

**Operation:** The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

\[(\text{PSW}) \leftarrow (\text{SP})\]
\[\text{SP} \leftarrow \text{SP} + 2\]

**Assembly Language Format:** POPF

**Object Code Format:** [ 11110011 ]

**States:**
- Onchip Stack: 9
- Offchip Stack: 13

---

3.13.73. PUSH — PUSH WORD

**Operation:** The specified operand is pushed onto the stack.

\[\text{SP} \leftarrow \text{SP} - 2\]
\[(\text{SP}) \leftarrow (\text{DEST})\]

**Assembly Language Format:** PUSH waop

**Object Code Format:** [ 110010aa ] [ waop ]

**States:**
- Onchip Stack: 8 + CEA
- Offchip Stack: 12 + CEA

---

**Flags Affected**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.74. PUSHF — PUSH FLAGS

**Operation:** The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

\[
\begin{align*}
\text{SP} & \leftarrow \text{SP} - 2 \\
(\text{SP}) & \leftarrow \text{PSW} \\
\text{PSW} & \leftarrow 0
\end{align*}
\]

**Assembly Language Format:** PUSHF

**Object Code Format:** \[
\begin{bmatrix} 11110010 \end{bmatrix}
\]

| Bytes: | 1 |
| States: | Onchip Stack: 8 |
|         | Offchip Stack: 12 |

<table>
<thead>
<tr>
<th>Flags Affected</th>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3.13.75. RET — RETURN FROM SUBROUTINE

**Operation:** The PC is popped off the top of the stack.

\[
\begin{align*}
\text{PC} & \leftarrow (\text{SP}) \\
\text{SP} & \leftarrow \text{SP} + 2
\end{align*}
\]

**Assembly Language Format:** RET

**Object Code Format:** \[
\begin{bmatrix} 11110000 \end{bmatrix}
\]

| Bytes: | 1 |
| States: | Onchip Stack: 12 |
|         | Offchip Stack: 16 |

<table>
<thead>
<tr>
<th>Flags Affected</th>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
3.13.76. RST — RESET SYSTEM

Operation: The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value (see section 2.15.2, "Reset Status"). Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW ← 0
PC ← 2080H

Assembly Language Format: RST

Object Code Format: [ 11111111 ]

Bytes: 1
States: 16

---

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3.13.77. SCALL — SHORT CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

SP ← SP − 2
(SP) ← PC
PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SCALL cadd

Object Code Format: [ 00101xxx ] [ disp-low ]

where xxx holds the three high-order bits of displacement.

Bytes: 2
States: Onchip Stack: 13
Offchip Stack: 16

---

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3.13.78. SETC — SET CARRY FLAG

**Operation:** The carry flag is set.

\[ C \leftarrow 1 \]

**Assembly Language Format:** SETC

**Object Code Format:** [ 11111001 ]

Bytes: 1
States: 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

3.13.79. SHL — SHIFT WORD LEFT

**Operation:** The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

\[
\text{Temp} \leftarrow \text{(COUNT)}
\]
\[
\text{do while Temp} \not= \text{0}
\]
\[
\begin{align*}
\text{C} & \leftarrow \text{High order bit of (DEST)} \\
\text{(DEST)} & \leftarrow (\text{DEST}) \times 2 \\
\text{Temp} & \leftarrow \text{Temp} - 1
\end{align*}
\]
\[
\text{end while}
\]

**Assembly Language Format:**

- SHL wreg,#count
  
  or

- SHL wreg,breg

**Object Code Format:** [ 00001001 ][ cnt/breg ][ wreg ]

Bytes: 3
States: 7 + No. of shifts performed

note: 0 place shifts take 8 states

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.80. SHLB — SHIFT BYTE LEFT

**Operation:** The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST)*2
    Temp ← Temp - 1
end _while

**Assembly Language Format:**

- SHLB breg,#count
  - or
- SHLB breg,breg

**Object Code Format:**

- [00011001][cnt/breg][breg]

- Bytes: 3
- States: 7 + No. of shifts performed
  - note: 0 place shifts take 8 states.

**Flags Affected**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>✓</td>
<td>↑</td>
<td>−</td>
</tr>
</tbody>
</table>
3.13.81. SHLL — SHIFT DOUBLE-WORD LEFT

Operation: The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

\[
\text{Temp} \leftarrow \text{(COUNT)} \\
\text{do while Temp} \not= 0 \\
\quad C \leftarrow \text{High order bit of (DEST)} \\
\quad (\text{DEST}) \leftarrow (\text{DEST})^2 \\
\quad \text{Temp} \leftarrow \text{Temp} - 1 \\
\text{end while}
\]

Assembly Language Format: 

- \text{SHLL } \text{i/reg,#count}
- \text{SHLL } \text{i/reg,breg}

Object Code Format: 

\[
[00001101] [\text{cnt/breg}] [\text{i/reg}]
\]

- Bytes: 3
- States: 7 + No. of shifts performed
  - note: 0 place shifts take 8 states.

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>√</td>
<td>?</td>
<td>√</td>
<td>√</td>
<td>↑</td>
<td>—</td>
</tr>
</tbody>
</table>

3-65
3.13.82. SHR — LOGICAL RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

\[
\text{Temp} \leftarrow (\text{COUNT}) \\
do \text{while} \text{Temp <> 0} \\
\quad C \leftarrow \text{Low order bit of (DEST)} \\
\quad (\text{DEST}) \leftarrow (\text{DEST}) / 2 \text{ where } / \text{ is unsigned division} \\
\quad \text{Temp} \leftarrow \text{Temp - 1} \\
\text{end_while}
\]

Assembly Language Format: 

SHR wreg,#count
or
SHR wreg,breg

Object Code Format: [ 00001000 ][ cnt/breg ][ wreg ]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected:

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>0</td>
<td>✓</td>
<td>0</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.83. SHRA — ARITHMETIC RIGHT SHIFT WORD

**Operation:** The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

\[ \text{Temp} \leftarrow (\text{COUNT}) \]
\[ \text{do while Temp } \gg 0 \]
\[ \quad C \leftarrow \text{Low order bit of (DEST)} \]
\[ \quad (\text{DEST}) \leftarrow (\text{DEST}) / 2 \text{ where } / \text{ is signed division} \]
\[ \quad \text{Temp} \leftarrow \text{Temp} - 1 \]
\[ \text{end while} \]

**Assembly Language Format:**

- SHRA wreg,#count
- SHRA wreg,breg

**Object Code Format:**

\[ \begin{array}{c|c}
00001010 & [ \text{cnt/breg }][ \text{ wreg }] \\
\end{array} \]

- Bytes: 3
- States: 7 + No. of shifts performed
  - note: 0 place shifts take 8 states.

**Flags Affected**

<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>/</td>
<td>/</td>
<td>0</td>
<td>-</td>
<td>/</td>
</tr>
</tbody>
</table>
3.13.84. SHARB — ARITHMETIC RIGHT SHIFT BYTE

**Operation:** The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

Temp ← (COUNT)
do while Temp <> 0
  C, = Low order bit of (DEST)
  (DEST) ← (DEST) / 2 where / is signed division
  Temp ← Temp − 1
end while

**Assembly Language Format:**

SHARB breg,#count
or
SHARB breg,breg

**Object Code Format:**

[ 00011010 ][ cnt/breg ][ breg ]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z N C V VT ST</td>
</tr>
<tr>
<td>✓ ✓ ✓ 0 − ✓</td>
</tr>
</tbody>
</table>
3.13.85. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where/is signed division
    Temp ← Temp - 1
end _ while

Assembly Language Format: SHRAL Ireg,#count
or
SHRAL Ireg,breg

Object Code Format: [ 00001110 ][ cnt/breg ][ Ireg ]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.86. **SHRB — LOGICAL RIGHT SHIFT BYTE**

**Operation:** The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

Temp ← (COUNT)
do while Temp <> 0
   C ← Low order bit of (DEST)
   (DEST) ← (DEST) / 2 where/is unsigned division
   Temp ← Temp - 1
end while

**Assembly Language Format:**

```
SHRB  breg,#count
or
SHRB  breg,breg
```

**Object Code Format:**

```
[ 00011000 ][ cnt/breg ][ breg ]
```

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

3-70
3.13.87. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

**Operation:** The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

Temp ← (COUNT)  
do while Temp <> 0  
    C ← Low order bit of (DEST)  
    (DEST) ← (DEST) / 2 where/is unsigned division  
    Temp ← Temp – 1  
end_ while

**Assembly Language Format:**
- SHRL lreg,#count
- SHRL lreg,breg

**Object Code Format:**
- [ 00001100 ] [ cnt/breg ] [ lreg ]

Bytes: 3  
States: 7 + No. of shifts performed  
note: 0 place shifts take 8 states.

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.88. SJMP — SHORT JUMP

**Operation:** The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of -1024 to +1023 inclusive.

\[
PC \leftarrow PC + \text{disp} \text{ (sign-extended to 16 bits)}
\]

**Assembly Language Format:** `SJMP cadd`

**Object Code Format:**

\[
\begin{array}{c}
00100xxx \\
\text{disp-low}
\end{array}
\]

where `xxx` holds the three high order bits of the displacement.

- **Bytes:** 2
- **States:** 8

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>

3.13.89. SKIP — TWO BYTE NO-OPERATION

**Operation:** Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

**Assembly Language Format:** `SKIP breg`

**Object Code Format:**

\[
\begin{array}{c}
00000000 \\
\text{breg}
\end{array}
\]

- **Bytes:** 2
- **States:** 4

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--</td>
</tr>
</tbody>
</table>
3.13.90. ST — STORE WORD

Operation: The value of the leftmost word operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format: SRC DST
ST wreg, waop

Object Code Format: [ 110000aa ][ waop ][ wreg ]
Bytes: 2 + BEA
States: 4 + CEA

Flags Affected:
<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

3.13.91. STB — STORE BYTE

Operation: The value of the leftmost byte operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format: SRC DST
STB breg, baop

Object Code Format: [ 110001aa ][ baop ][ breg ]
Bytes: 2 + BEA
States: 4 + CEA

Flags Affected:
<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3.13.92. SUB (Two Operands) — SUBTRACT WORDS

**Operation:** The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

\[(\text{DEST}) \leftarrow (\text{DEST}) - (\text{SRC})\]

**Assembly Language Format:**
```
DST  SRC
SUB  wreg, waop
```

**Object Code Format:**
```
[ 011010aa ][ waop ][ wreg ]
```

Bytes: 2 + BEA
States: 4 + CEA

---

Flags Affected:
```
<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>↑</td>
<td>−</td>
</tr>
</tbody>
</table>
```

3.13.93. SUB (Three Operands) — SUBTRACT WORDS

**Operation:** The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

\[(\text{DEST}) \leftarrow (\text{SRC1}) - (\text{SRC2})\]

**Assembly Language Format:**
```
DST  SRC1  SRC2,
SUB  wreg, wreg, waop
```

**Object Code Format:**
```
[ 010010aa ][ waop ][ Dwreg ][ Swreg ]
```

Bytes: 3 + BEA
States: 5 + CEA

---

Flags Affected:
```
<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>↑</td>
<td>−</td>
</tr>
</tbody>
</table>
```
### 3.13.94. SUBB (Two Operands) — SUBTRACT BYTES

**Operation:** The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

\[(\text{DEST}) \leftarrow (\text{DEST}) - (\text{SRC})\]

**Assembly Language Format:**

```
DST SRC
SUBB breg, baop
```

**Object Code Format:**

```
[ 011110aa ][ baop ][ breg ]
```

- **Bytes:** 2 + BEA
- **States:** 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>

### 3.13.95. SUBB (Three Operands) — SUBTRACT BYTES

**Operation:** The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

\[(\text{DEST}) \leftarrow (\text{SRC1}) - (\text{SRC2})\]

**Assembly Language Format:**

```
DST SRC1 SRC2
SUBB breg, Sbreg baop
```

**Object Code Format:**

```
[ 010110aa ][ baop ][ Sbreg ][ Dbreg ]
```

- **Bytes:** 3 + BEA
- **States:** 5 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>✓</td>
</tr>
</tbody>
</table>
3.13.96. SUBC — SUBTRACT WORDS WITH BORROW

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

\[(DEST) \leftarrow (DEST) - (SRC) - (1-C)\]

Assembly Language Format: \(\text{DST SRC SUBC wreg, waop}\)

Object Code Format: [101010aa][waop][wreg]

Bytes: 2 + BEA
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>↓</td>
</tr>
</tbody>
</table>

3.13.97. SUBCB — SUBTRACT BYTES WITH BORROW

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

\[(DEST) \leftarrow (DEST) - (SRC) - (1-C)\]

Assembly Language Format: \(\text{DST SRC SUBCB breg, baop}\)

Object Code Format: [101110aa][baop][breg]

Bytes: 2 + BEA
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td>↓</td>
</tr>
</tbody>
</table>
3.13.98. TRAP — SOFTWARE TRAP

**Operation:** This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

\[ SP \leftarrow SP - 2 \]
\[ (SP) \leftarrow PC \]
\[ PC \leftarrow (2010H) \]

**Assembly Language Format:** This instruction is not supported by revision 1.0 of the 8096 assembly language.

**Object Code Format:**

<table>
<thead>
<tr>
<th>Bytes:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>States:</td>
<td>Onchip Stack: 21</td>
</tr>
<tr>
<td></td>
<td>Offchip Stack: 24</td>
</tr>
</tbody>
</table>

<p>| Flags Affected |
|---|---|---|---|---|</p>
<table>
<thead>
<tr>
<th>Z</th>
<th>N</th>
<th>C</th>
<th>V</th>
<th>VT</th>
<th>ST</th>
</tr>
</thead>
</table>

3.13.99. XOR — LOGICAL EXCLUSIVE-OR WORDS

**Operation:** The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

\[ (DEST) \leftarrow (DEST) \text{ XOR (SRC)} \]

**Assembly Language Format:**

<table>
<thead>
<tr>
<th>DST</th>
<th>SRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>wreg, waop</td>
</tr>
</tbody>
</table>

**Object Code Format:**

<table>
<thead>
<tr>
<th>Bytes:</th>
<th>2 + BEA</th>
</tr>
</thead>
<tbody>
<tr>
<td>States:</td>
<td>4 + CEA</td>
</tr>
</tbody>
</table>

| Flags Affected |
|---|---|---|---|
| Z | N | C | V | VT | ST |
| ✓ | ✓ | 0 | 0 | - | - |
3.13.100. XORB — LOGICAL EXCLUSIVE-OR BYTES

**Operation:** The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

\[(\text{DEST}) \leftarrow (\text{DEST}) \text{ XOR} (\text{SRC})\]

**Assembly Language Format:**

DST SRC
XORB breg, baop

**Object Code Format:**

\[\begin{array}{c}
100101aa \\
[\text{baop}][\text{breg}]
\end{array}\]

Bytes: 2 + BEA
States: 4 + CEA

<table>
<thead>
<tr>
<th>Flags Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

3-78
CHAPTER 4
MCS®-96 HARDWARE DESIGN INFORMATION

4.0. HARDWARE INTERFACING OVERVIEW

This section of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this section a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

4.1. REQUIRED HARDWARE CONNECTIONS

Although the 8096 is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 4-5 shows the connections that are needed for a single-chip system.

4.1.1. Power Supply Information

Power for 8096 flows through 6 pins; one VCC pin, two VSS pins, one VREF (analog VCC), one ANGND (Analog VSS), and one VPD (V Power Down) pin. All six of these pins must be connected to the 8096 for normal operation. The VCC pin, VREF pin and VPD pin should be tied to 5 volts. When the analog to digital converter is being used it may be desirable to connect the VREF pin to a separate power supply, or at least a separate power supply line.

The two VSS pins should be connected together with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between VSS1 and VSS2. The 2 VSS pins and the ANGND pin should all be nominally at 0 volts. The maximum current drain of the 8096 is around 200mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The VREF pin supplies 5 volts to the analog circuitry and the ANGND pin is the ground for this section of the chip. More information on the analog power supply is in section 4.3.1.

4.1.2. Other Needed Connections

Several of the pins on the 8096 are used to configure the mode of operation. In normal operation the following pins should be tied directly to the indicated power supply.

<table>
<thead>
<tr>
<th>PIN</th>
<th>POWER SUPPLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI</td>
<td>VCC</td>
</tr>
<tr>
<td>TEST</td>
<td>VCC</td>
</tr>
<tr>
<td>EA</td>
<td>VCC (to allow internal execution)</td>
</tr>
<tr>
<td>VSS</td>
<td>VCC (to force external execution)</td>
</tr>
</tbody>
</table>

Figure 4-1. 8096 Oscillator Circuit

Figure 4-2. Crystal Oscillator Circuit
Although the EA pin has an internal pulldown, it is best to tie this pin to the desired level if it is not left completely disconnected. This will prevent induced noise from disturbing the system.

4.1.3. Oscillator Information

The 8096 requires a clock source to operate. This clock can be provided to the chip through the XTAL1 input or the on-chip oscillator can be used. The frequency of operation is from 6.0 MHz to 12 MHz.

The on-chip circuitry for the 8096 oscillator is a single stage linear inverter as shown in Figure 4-1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 4-2. In this application, the crystal is being operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

The crystal specifications and capacitance values (C1 and C2 in Figure 4-2) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

To drive the 8096 with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 4-3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels. It is important that the minimum high and low times are met.

There is no specification on rise and fall times, but they should be reasonably fast (on the order of 30 nanoseconds) to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096 internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4-4 shows the waveforms of the major internal timing signals.

![Figure 4-3. External Clock Drive](image)

![Figure 4-4. Internal Timings](image)
4.1.4. Reset Information

In order for the 8096 to function properly it must be reset. This is done by holding the reset pin low for at least 2 state times after the power supply is within tolerance, the oscillator has stabilized, and the back-bias generator has stabilized. Typically, the back-bias generator requires one millisecond to stabilize.

There are several ways of doing this, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 1 to 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of VCC is fast and the total reset time is less than around 50 milliseconds. It also may not work if the reset pin is to be used to reset other parts on the board. An 8096 with the minimum required connections is shown in Figure 4-5.

The 8096 RESET pin can be used to allow other chips on the board to make use of the watchdog timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open-collector output. The reset pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the RESET low duration is only two state times. If this is done, it is possible that the 8096 will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to RESET cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull down more slowly than normal. It will continue to pull down until the 8096 is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. Several circuit examples are shown in Figure 4-6.

4.1.5. Sync Mode

If RESET is brought high at the same time as or just after the rising edge of XTAL1, the part will start executing the 10 state time RST instruction exactly 6'12 XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 4-7. It should be noted that parts that start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.

---

[Figure 4-5. Minimum Hardware Connections]

NOTES: 1. THESE CAPACITORS ARE NEEDED ONLY IF A TO D IS USED.
2. VREF & ANGND MAY BE CONNECTED TO THE SAME TRACES AS THE DIGITAL POWER SUPPLY IF THE A TO D IS NOT USED.
4.1.6. Disabling the Watchdog Timer

The watchdog timer will pull the RESET pin low when it overflows. If the pin is being externally held above the low going threshold, the pull-down transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or RESET must be held high indefinitely. Just resetting the watchdog timer will not clear the flip-flop which keeps the RESET pull-down on.

The pull-down is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current
may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the watchdog timer over-ridden for more than a second or two.

Whenever the reset pin is being pulled high while the pull-down is on, it should be through a resistor that will limit the voltage on \texttt{RESET} to 2.5 volts and the current through the pin to 40 milliamps. Figure 4-8 shows a circuit which will provide the desired results. Using the LED will provide the additional benefit of having a visual indicator that the part is trying to reset itself, although this circuit only works at room temperature and \texttt{VCC} = 5 Volts.

If it is necessary to disable the watchdog timer for more than a brief test the software solution of never initiating the times should be used. See Section 2.14.

4.1.7. Power Down Circuitry

Battery backup can be provided on the 8096 with a 1 mA current drain at 5 volts. This mode will hold locations \texttt{0FH} through \texttt{OFFH} valid as long as the power to the \texttt{VPD} pin remains on. The required timings to put the part into power-down and an overview of this mode are given in section 2.4.2.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096 until it has completed its reset operation.

4.2. DRIVE AND INTERFACE LEVELS

There are 5 types of I/O lines on the 8096. Of these, 2 are inputs and 3 are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as \texttt{XTAL1} and \texttt{RESET}, may have slightly different characteristics. These pins are discussed in section 4.1.

While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the latest version of the 8096 Data Sheet.

4.2.1. Quasi-Bidirectional Ports

The quasi-bidirectional port is both an input and an output port. It has three states, low impedance current sink, low impedance current source, and high impedance current source. As a low impedance current sink, the pin has a specification of sinking up to around .4 milliamps, while staying below 0.45 volts. The pin is placed in this condition by writing a ‘0’ to the SFR (Special Function Register) controlling the pin.

When a ‘1’ is written to the SFR location controlling the pin, a low impedance current source is turned on for one state time, then it is turned off and the depletion pull-up holds the line at a logical ‘1’ state. The low-impedance pull-up is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pull-up. The configuration of a quasi-bidirectional port pin is shown in Figure 4-9.

While the depletion mode pull-up is the only device on, the pin may be used as an input with a leakage of around

![Figure 4-8. Disabling the WDT](image-url)
100 microamps from 0.45 volts to VCC. It is ideal for use with TTL or CMOS chips and may even be used directly with switches, however if the switch option is used certain precautions should be taken. It is important to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could prevent logical operations on these pins while they are being used as inputs.

4.2.2. Quasi-Bidirectional Hardware Connections

When using the quasi-bidirectional ports as inputs tied to switches, series resistors should be used if the ports will be written to internally after the part is initialized. Every time any quasi-bidirectional pin is written from a zero to a one, the low impedance pull-up is turned on. If many of the pins are tied directly to ground, a large current spike will be generated when all of these low impedance devices are turned on at once.

For this reason, a series resistor is recommended to limit the current to a maximum of 0.2 milliamps per pin. If several pins are connected to a common ground through switches, it should be sufficient to limit the current through the common ground to 0.2 milliamps times the maximum number of pins that could be switched to ground. Many switches require a minimum amount of current flow through them to keep them clean. This could cause problems in long term reliability if it is not considered when designing a system.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pull-up resistor is recommended to reduce the possibility of noise glitches and to decrease the rise time of the line. On extremely long lines that are handling slow signals a capacitor may be helpful in addition to the resistor to reduce noise.

4.2.3. Input Ports, Analog and Digital

The high impedance input ports on the 8096 have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pf. The Port 0 pins have

Figure 4-9. Quasi-Bidirectional Port
an additional function when the A to D converter is being used. These pins are the input to the A to D converter, and as such, are required to provide current to the comparator when a conversion is in process. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See section 4.2.1.

4.2.4. Open Drain Ports

Ports 3 and 4 on the 8096 are open drain ports. There is no pull-up when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to ports 3 and 4 and the Bus pins is shown in Figure 4-10.

When Ports 3 and 4 are to be used as inputs, or as Bus pins, they must first be written with a '1', this will put the ports in a high impedance mode. When they are used as outputs, a pull-up resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pull-up current to the pin must be less than this. A 15k pull-up resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.

4.2.5. HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are: TXD, RXD (in mode 0), PWM, CLKOUT, ALE, BHE, RD, and WR. The bus pins have 3 states: output high, output low, and high impedance input. As a high output, the pins are specified to source around 200 μA to 2.4 volts, but the pins can source on the order of ten times that value in order to provide fast rise times. When used as a low output, the pins can sink around 2 mA at .45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 4-10 shows the internal configuration of a bus pin.

Figure 4-10. Bus and Port 3 and 4 Pins
4.3. ANALOG INTERFACE

Interfacing the 8096 to analog signal can be done in several ways. If the 8096 needs to measure an analog signal the A to D converter can be used. Creation of analog outputs can be done with either the PWM output or the HSO unit.

4.3.1. Analog Inputs

The 8096 can have 8 analog inputs and can convert one input at a time into a digital value. Each conversion takes 42 microseconds with a 12 MHz signal on XTAL1. The input signal is applied to one of the Port O/Analog Channel inputs. Since there is no sample and hold on the A to D, the input signal must remain constant over the sampling period.

When a conversion takes place, the 8096 compares the external signal to that of its internal D to A. Based on the result of the comparison it adjusts the D to A and compares again. Each comparison takes 8 state times and requires the input to the comparator to be charged up. 20 comparisons are made during a conversion, two times for each bit of resolution. An additional 8 states are used to load and store values. The total number of state times required is 168 for a 10-bit conversion. Attempting to do other than a 10-bit conversion is not recommended.

Since the capacitance of the comparator input is around 0.5pf, the sample and hold circuit must be able to charge a 10pf (20*0.5pf) capacitor without a significant voltage change. To keep the effect of the sample and hold circuit below ±½ lsb on a 10-bit converter, the voltage on the sample and hold circuit may vary no more than 0.05% (1/2048).

The effective capacitance of the sample and hold must, therefore, be at least 20000pf or 0.02 uf. If there is external leakage on the capacitor, its value must be increased to compensate for the leakage. At 10µA leakage, 2.5 mV (5/2048) will be lost from a 0.17 uf capacitor in 42 µS. The capacitor connected externally to the pin should, therefore, be at least 0.2 uf for best results. If the external signal changes slowly relative to 42 µS, then a larger capacitor will work well and also filter out unwanted noise.

The converter is a 10-bit, successive approximation, ratiometric converter, so the numerical value obtained from the conversion will be:

\[ 1023 \times \frac{(V_{IN} - \text{ANGND})}{(V_{REF} - \text{ANGND})} \]

It can be seen that the power supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. VREF should be well regulated and used only for the A to D converter. If ratiometric information is desired, VREF can be connected to VCC, but this should be done at the power supply not at the chip. It needs to be able to source around 15 milliamps. Bypass capacitors should be used between VREF and ANGND. ANGND should be within a tenth of a volt of VSS and VREF should be within a few tenths of a volt of VCC. A 0.01 uf capacitor should be connected between the ANGND and ANOND pins.

---

**Figure 4-11. D/A Buffer Block Diagram**

**SUGGESTED CIRCUIT FOR NON-CRITICAL APPLICATIONS**

- 8096
- CD4049
- R and C are chosen for best filtering at the user's frequency
- ANALOG OUTPUT
- POWER AMP (OPTIONAL)
- FILTER (PASSIVE OR ACTIVE)
- SWING RAIL TO RAIL
- BUFFER TO MAKE OUTPUT
- HSO OR PWM
- 8096
- HIGH IMPEDANCE AMP
- ANALOG OUTPUT
- RAND C ARE CHOSEN FOR BEST FILTERING AT THE USER'S FREQUENCY

4-8
VBB pins to reduce the noise on VBB and provide the highest possible accuracy. Figure 4-5 shows all of these connections.

4.3.2. Analog Output Suggestions
Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will generate a rectangular pulse train that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 4-11. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64 uS is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

4.4. I/O TIMINGS
The I/O pins on the 8096 are sampled and changed at specific times within an instruction cycle. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 nanoseconds using the information in this section is not recommended.

4.4.1. HSO Outputs
Changes in the HSO lines are synchronized to Timer 1. All of the external HSO lines due to change at a certain value of a timer will change just prior to the incrementing of Timer 1. This corresponds to an internal change during Phase C, every eight state times. From an external perspective the HSO pin should change around the rising edge of CLKOUT and be stable by its falling edge. Internal events can occur anytime during the 8 state time window.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

4.4.2. HSI Input Sampling
The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1

full state time to guarantee that it is recognized. This restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. If the events occur on different pins they will always be recorded, regardless of the time difference. The 8 state time window, (ie. the amount of time during which Timer 1 remains constant), is stable to within about 20 nanoseconds. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

4.4.3. Standard I/O Port Pins
Port 0 is different from the other digital ports in that it is actually part of the A to D converter. The port is sampled once every 8 state times, the same frequency at which the comparator is charged-up during an A to D conversion. This 8 state times counter is not synchronized with Timer 1. If this port is used the input signal on the pin must be stable 8 state times prior to reading the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from ‘0’ to a ‘1’ the low impedance pull-up will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drain, their structure is different when they are used as part of the bus. See Section 2.12.4.

4.5. SERIAL PORT TIMINGS
The serial port on the 8096 was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096 uses a divide by 3, the serial port on the 8096 had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096 so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK, because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to 1/6 that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 2.11.4 discusses programming the baud rate generator.

4.5.1. Mode 0
Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 4-12 shows the waveforms and timing. Note that the port starts functioning when a ‘1’ is written
to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096 by simply adding shift registers. A schematic of a typical circuit is shown in Figure 4-13. This circuit inverts the data coming in, so it must be re-inverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

Figure 4-12. Serial Port Timings in Mode 0

Figure 4-13. Mode 0 Serial Port Example
4.5.2. Mode 1 Timings

Mode 1 operation of the serial port makes use of 10-bit data packages, a start bit, 8 data bits and a stop bit. The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI

Figure 4-14. Bus Signal Timings
is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

Caution should be used when using the serial port to connect more than two devices in half-duplex, (ie. one wire for transmit and receive). If the receiving processor does not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be squashed. This could cause a problem for other devices listening on the link.

4.5.3. Mode 2 and 3 Timings
Modes 2 and 3 operate in a manner similar to that of mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit.

**Timings The Memory System Must Meet**

- **TLLYH** — ALE low to READY high: Maximum time after ALE falls until READY is brought high to ensure no more wait states. If this time is exceeded unexpected wait states may result. Nominally 1 Tosc + 3 Tosc* number of wait states desired.

- **TLLYV** — ALE low to READY VALID: Maximum time after ALE falls until READY must be valid. If this time is exceeded the part could malfunction necessitating a chip reset. Nominally 2 Tosc periods.

- **TYLYH** — READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096 dynamic nodes which hold the current instruction may ‘forget’ how to finish the instruction.

- **TAVDV** — ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096 outputs a valid address. Nominally, a maximum of 5 Tosc periods.

- **TRLDV** — READ low to DATA valid: Maximum time that the memory has to output data after READ goes low. Nominally, a maximum of 3 Tosc periods.

- **TRXDZ** — READ not low to DATA float: Time after READ is no longer low until the memory must float the bus. The memory signal can be removed as soon as READ is not low, and must be removed within the specified maximum time. Nominally, a maximum of 1 Tosc period.

**Timings the 8096 Will Provide**

- **TCHCH** — CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 Tosc average, but individual periods could vary by a few nanoseconds.

- **TCHCL** — CLKOUT high to CLKOUT low: Nominally 1 Tosc period.

- **TCLLH** — CLKOUT low to ALE high: A help in deriving other timings, typically plus or minus 5 to 10 ns.

- **TLLCH** — ALE low to CLKOUT high: Used to derive other timings, nominally 1 Tosc period.

- **TLHLL** — ALE high to ALE low: ALE pulse width. Useful in determining ALE rising edge to ADDRESS valid time. Nominally 1 Tosc period.

- **TAVLL** — ADDRESS valid to ALE low: Length of time ADDRESS is valid before ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

- **TLLAX** — ALE low to ADDRESS invalid: Length of time ADDRESS is valid after ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

- **TLLRL** — ALE low to READ or WRITE low: Length of time after ALE falls before RD or WR fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 Tosc period.

- **TRLRH** — READ low to READ high: RD pulse width, nominally 1 Tosc period.

- **TRHLH** — READ high to ALE high: Time between RD going inactive and next ALE, also used to calculate time between RD inactive and next ADDRESS valid. Nominally 1 Tosc period.

- **TWTWH** — WRITE low to WRITE high: Write pulse width, nominally 2 Tosc periods.

- **TQVWX** — OUTPUT valid to WRITE not low: time that the OUTPUT data is valid before WR starts to go high. Nominally 2 Tosc periods.

- **TWXQX** — WRITE not low to OUTPUT not valid: Time that the OUTPUT data is valid after WR starts to rise. Nominally 1 Tosc period.

- **TWXLH** — WRITE not low to ALE high: Time between write starting to rise and next ALE, also used to calculate the time between WR starting to rise and next ADDRESS valid. Nominally 2 Tosc periods.

---

**Figure 4-15. Timing Specification Explanations**

---

4-12
rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see section 2.11).

4.6. BUS TIMING AND MEMORY INTERFACE

4.6.1. Bus Functionality
The 8096 has a multiplexed (address/data) 16 bit bus. There are control lines provided to demultiplex the bus (ALE), indicate reads or writes (RD, WR), indicate if the access is for an instruction (INST), and separate the bus into high and low bytes (BHE, AD0). Section 2.3.5 contains an overview of the bus operation.

4.6.2. Timing Specifications
Figure 4-14 shows the timing of the bus signals and data lines. Since this is a new part, the exact timing specifications are subject to change, please refer to the latest 8096 data sheet to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 4-15.

4.6.3. READY Line Usage
When the processor has to address a memory location that cannot respond within the standard specifications it is necessary to use the READY line to generate wait states. When the READY line is held low the processor waits in a loop for the line to come high. There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls (or the TYVCL before CLKOUT falls) or the processor could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated. To ensure that only one wait state is inserted it is necessary to bring READY high TLLYH after the falling edge of ALE.

4.6.4. INST Line Usage
The INST (Instruction) line is high during the output of an address that is for an instruction stream fetch. It is low during the same time for any other memory access. At any other time it is not valid. This pin is not present on the 48-pin versions. The INST signal can be used with a logic analyzer to debug a system. In this way it is possible to determine if the fetch was for instructions or data, making the task of tracing the program much easier.

4.6.5. Address Decoding
The multiplexed bus of the 8096 must be demultiplexed before it can be used. This can be done with 2 74LS373 transparent latches. As explained in section 2.3.5, the latched address signal will be referred to as MA0 through MA15. (Memory Address), and the data lines will be called MD0 through MD15, (Memory Data).

Since the 8096 can make accesses to memory for either bytes or words it is necessary to have a way of determining the type of access desired. The BHE and MA0 lines are used for this purpose. BHE must be latched, as it is valid only when the address is valid. The memory system is typically set up as 32K by 16, instead of 64K by 8. When the BHE line is low, the upper byte is enabled. When MA0 is low, the lower byte is enabled when MA0 is high BHE will be low, and the upper byte is enabled.

When external RAM and EPROM are both used in the system the control logic can be simplified a little to some of the addresses. The 8096 will always output BHE to indicate if a read is of the high byte or the low byte, but it discards the byte it is not going to use. It is therefore possible to use the BHE and MA0 lines only to control

---

Figure 4-16. Memory Wiring Example—EPROM Only System
memory writes, and to ignore these lines during memory reads. Figure 4-16 and 4-17 show block diagrams of two memory systems, an external EPROM only system and a RAM/ROM system.

4.6.6. System Verification Example
To verify that a system such as the one in Figure 4-17 will work with the 8096, it is necessary to check all of the timing parameters. Let us examine this system one parameter at a time using the proposed 8096 specifications. These specifications are subject to change, refer to the latest 8096 data sheet for the current specifications.

The timings of signals that the processor and memory use are affected by the latch and buffer circuitry. The timings of the signal provided by the processor are delayed by various amounts of time. Similarly, the signals coming back from the memory are also delayed. The calculations involved in verifying this system follow:

**Address Valid Delay — 20 nanoseconds**

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18ns after Address is valid or 30ns after ALE is high. Since the signal may be limited by either the ALE timing or the Address timing, these two cases must be considered.

If Limited by ALE:
- Minimum ALE pulse width = Tosc-10 (TLHLL)
- Minimum Addr set-up to ALE falling = Tosc-20 (TAVLL)

Therefore ALE could occur 10 ns before Address valid.

Total delay from 8096 Address stable to MA (Memory Address) stable would be:

\[
\text{ALE delay from address} - 10 \\
74LS373 \text{ clock to output} = 30 \\
\text{20 nanoseconds}
\]

If Limited by Address Valid:

- 74LS373 Data Valid to Data Output = 18 nanoseconds

In the worst case, the delay in Address valid is controlled by ALE and has a value of 20 nanoseconds.

![Figure 4-17. RAM/ROM Memory System](image-url)
Delay of Data Transfer to/from Processor — 12 nanoseconds

The RD low to Data valid specification (TRLDV) is 3 Tosc-50, (200 ns at 12 MHz). The 74LS245 is enabled by RD and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled except during a read, so there is no enable delay to consider for write operations.

The Data In to Data Out delay of the 74LS245 is 12 ns.

Delay of WR signal to memory — 15 nanoseconds

Latched BHE is delayed by the inverter on ALE and the 74LS74.

74LS04 delay (Output low to high) = 22
74LS74 delay (Clock to Output) = 40
Delay of Latched BHE from ALE falling = 62 nanoseconds

The 74LS74 requires data valid for 20 ns prior to the clock, the 8096 will have BHE stable Tosc-20 ns (TAVLL, 63 ns at 12 MHz) prior to ALE falling. There is no problem here.

MAO is valid prior to ALE falling, since the 20 ns Address Delay is less than TAVLL.

WR will fall no sooner than Tosc-20 ns (TLLRL, 63 ns at 12 MHz) after ALE goes low. It will therefore be valid just after the Latched BHE is valid, so it is the controlling signal.

WR High and WR Low are valid 15 ns after MAO, Latched BHE and WR are valid. Since WR is the last signal to go valid, the delay of WR (High and Low) to memory is 15 ns.

Delay Summary — Address Delay = 20 ns
Data Delay = 12 ns
WR Delay = 15 ns
RD Delay = 0 ns

Read low to Data in;

\[
\begin{array}{ll}
\text{TRLDV} & : 200.0 \text{ ns maximum} \\
\text{RD Delay} & : - 00.0 \text{ ns maximum} \\
\text{Data Delay} & : - 12.0 \text{ ns maximum} \\
\end{array}
\]

Provided by System:

Address valid to Control;

\[
\begin{array}{ll}
\text{TLLRL} & : 63.3 \text{ ns minimum} \\
\text{TAVLL} & : 63.3 \text{ ns minimum} \\
\text{Address Delay} & : - 20.0 \text{ ns maximum} \\
\text{WR Delay} & : - 00.0 \text{ ns maximum (no spec)} \\
\end{array}
\]

Write Pulse Width;

\[
\begin{array}{ll}
\text{TWLWH} & : 151.6 \text{ ns minimum} \\
\text{Rising WR Delay} & : - 15.0 \text{ ns maximum} \\
\text{Falling WR Delay} & : - 00.0 \text{ ns minimum (no spec)} \\
\end{array}
\]

Data Setup to WR rising;

\[
\begin{array}{ll}
\text{TQVWX} & : 136.6 \text{ ns minimum} \\
\text{Data Delay} & : - 12.0 \text{ ns maximum} \\
\text{WR Delay} & : - 00.0 \text{ ns minimum (no spec)} \\
\end{array}
\]

Data Hold after WR;

\[
\begin{array}{ll}
\text{TXWQX} & : 58.3 \text{ ns minimum} \\
\text{Data Delay} & : 0.0 \text{ ns minimum (no spec)} \\
\text{WR Delay} & : - 15.0 \text{ ns maximum} \\
\end{array}
\]

The two memory devices which are expected to be used most often with the 8096 are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

2764 Tac

(Address valid to Output) < Address valid to Data in
250 ns < 354 ns O.K.

2764 Toe

(Output Enable to Output) < Read low to Data in
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the RD (OE) line. If there are delays in these signals the delays must be added to the 2764’s timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in
200 ns < 354 ns O.K.

2128-20 Toe < Read low to Data in
65 ns < 188 ns O.K.

Characteristics of a 12 MHz 8096 system with latches:

Required by system:

Address valid to Data in;

\[
\begin{array}{ll}
\text{TAVDV} & : 386.6 \text{ ns maximum} \\
\text{Address Delay} & : - 20.0 \text{ ns maximum} \\
\text{Data Delay} & : - 12.0 \text{ ns maximum} \\
\end{array}
\]

354.6 ns maximum
The write calculations are a little more involved, but still straight-forward.

2128 Twp (Write Pulse) < Write Pulse Width
   100 ns < 146 ns  O.K.

2128 Tds (Data Setup) < Data Setup to WR rising
   65 ns < 124 ns  O.K.

2128 Tdh (Data Hold) < Data Hold after WR
   0 ns < 43 ns

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 4-17. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

4.6.7. I/O Port Reconstruction

When a single-chip system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 4-18 provides this function on the iSBE-96 emulator board. It can be attached to any 8096 system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when 1FFEH or 1FFFH are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096. The 'reset' line is used to set the ports to all 1's when the 8096 is reset. It should be noted that the voltage and current characteristics of this port will differ from those of the 8096, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at 1FFEH or 1FFFH. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

4.7. NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject...
to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the pc board to find itself in the path of electrostatic discharges. Glitches and noise on the pc board can cause the processor to act unpredictably, usually by changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096 has a watchdog timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

Many hardware solutions exist for keeping pc board noise to a minimum. Ground planes, gridded ground and VCC structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper pc board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Intel Application Note AP-125, "Designing Microcontroller Systems For Noisy Environments."

4.8. PACKAGING PINOUTS AND ENVIRONMENT

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without ROM, and with or without an A to D converter. A summary of the available options is shown in Figure 4-19.

The 48-pin versions are available in a 48-pin DIP (Dual In-Line) package, in either ceramic or plastic.

The 68-pin versions are available in a ceramic pin grid array, and a plastic flatpack. A plastic pin grid array will be available in the near future.

<table>
<thead>
<tr>
<th>ROMLESS</th>
<th>WITH ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>68-pin 48-pin</td>
<td>68-pin 48-pin</td>
</tr>
<tr>
<td>Without A to D</td>
<td>8096 8094 8396 8394</td>
</tr>
<tr>
<td>With A to D</td>
<td>8097 8095 8397 8395</td>
</tr>
</tbody>
</table>

Figure 4-19. The MCS®-96 Family of Products
16-BIT MICROCONTROLLERS

- 839X: an 809X with 8K Bytes of On-chip ROM
  - High Speed Pulse I/O
  - 10-bit A/D Converter
  - 8 Interrupt Sources
  - Pulse-Width Modulated Output
  - Four 16-bit Software Timers
  - 232 Byte Register File
  - Memory-to-Memory Architecture
  - Full Duplex Serial Port
  - Five 8-bit I/O Ports
  - Watchdog Timer

The MCS96 family of 16-bit microcontrollers consists of 8 members, all of which are designed for high-speed control functions.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0 μsec and a 16 x 16-bit multiply or 32/16-bit divide in 6.5 μsec. Instruction execution times average 1 to 2 μsec in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095, 8395, 8097 and 8397.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.

Figure 1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)
Figure 1 shows a block diagram of the MCS-96 parts, generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM. The MCS-96 numbering system is shown below:

<table>
<thead>
<tr>
<th>OPTIONS</th>
<th>68 PIN</th>
<th>48 PIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIGITAL I/O</td>
<td>ROMLESS</td>
<td>8096</td>
</tr>
<tr>
<td></td>
<td>ROM</td>
<td>8396</td>
</tr>
<tr>
<td>ANALOG AND DIGITAL I/O</td>
<td>ROMLESS</td>
<td>8097</td>
</tr>
<tr>
<td></td>
<td>ROM</td>
<td>8397</td>
</tr>
</tbody>
</table>

Figures 2, 3 & 4 show the pinouts for the 48- and 68-pin packages. The 48-pin version is offered in Dual-In-Line package while the 68-pin version comes in a Flat-pack and a Pin Grid Array.
### Figure 4. Pin Grid Array

#### Note

1. When the pin grid array package is mounted on the PC board, the pins are numbered counterclockwise as seen from the component side of the board, just like the flatpack when it's mounted in the contactor. Consequently, the PC board layout for pin grid array is compatible with the flat pack in a contactor except for the footprint size. The pin functions (from -1 to -68) on both packages are identical. Refer to Intel's Microcontroller handbook for mechanical dimensions on these packages.
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Operation (Note 1)</th>
<th>Flags</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD/ADDB</td>
<td>2</td>
<td>D ← D + A</td>
<td>Z N C V VT ST</td>
<td></td>
</tr>
<tr>
<td>ADD/ADDB</td>
<td>3</td>
<td>D ← B + A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDC/ADDCB</td>
<td>2</td>
<td>D ← D + A + C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB/SUBB</td>
<td>2</td>
<td>D ← D - A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB/SUBB</td>
<td>3</td>
<td>D ← B - A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBC/SUBCB</td>
<td>2</td>
<td>D ← D - A + C - 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP/CMPB</td>
<td>2</td>
<td>D - A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL/MULU</td>
<td>2</td>
<td>D, D + 2 ← D * A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL/MULU</td>
<td>3</td>
<td>D, D + 2 ← B * A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULB/MULUB</td>
<td>2</td>
<td>D, D + 1 ← D * A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULB/MULUB</td>
<td>3</td>
<td>D, D + 1 ← B * A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV/DIVU</td>
<td>2</td>
<td>D ← (D, D + 2)/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVB/DIVUB</td>
<td>2</td>
<td>D ← (D, D + 1)/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND/ANDB</td>
<td>2</td>
<td>D ← D and A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND/ANDB</td>
<td>3</td>
<td>D ← B and A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR/ORB</td>
<td>2</td>
<td>D ← D or A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XOR/XORB</td>
<td>2</td>
<td>D ← D (excl. or) A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD/LDB</td>
<td>2</td>
<td>D ← A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST/STB</td>
<td>2</td>
<td>A ← D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDBSE</td>
<td>2</td>
<td>D ← A; D + 1 ← SIGN(A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDBZSE</td>
<td>2</td>
<td>D ← A; D + 1 ← 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSH</td>
<td>1</td>
<td>SP ← SP - 2; (SP) A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>1</td>
<td>A ← (SP); SP ← SP + 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSHF</td>
<td>0</td>
<td>SP ← SP - 2; (SP) ← PSW; PSW ← 0000H</td>
<td>0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>POPF</td>
<td>0</td>
<td>PSW ← (SP); SP ← SP + 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SJMP</td>
<td>1</td>
<td>PC ← PC + 11-bit offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LJMP</td>
<td>1</td>
<td>PC ← PC + 16-bit offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR(indirect)</td>
<td>1</td>
<td>PC ← (A)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCALL</td>
<td>1</td>
<td>SP ← SP - 2; (SP) ← PC; PC ← PC + 11-bit offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCALL</td>
<td>1</td>
<td>SP ← SP - 2; (SP) ← PC; PC ← PC + 16-bit offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>0</td>
<td>PC ← (SP); SP ← SP + 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J(conditional)</td>
<td>1</td>
<td>PC ← PC + 8-bit offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JC</td>
<td>1</td>
<td>Jump if C = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JNC</td>
<td>1</td>
<td>Jump if C = 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.
Table 3-2. Instruction Summary

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operands</th>
<th>Operation (Note 1)</th>
<th>Flags</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>JE</td>
<td>1</td>
<td>Jump if Z = 1</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JNE</td>
<td>1</td>
<td>Jump if Z = 0</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JGE</td>
<td>1</td>
<td>Jump if N = 0</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JLT</td>
<td>1</td>
<td>Jump if N = 1</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JGT</td>
<td>1</td>
<td>Jump if N = 0 and Z = 0</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JLE</td>
<td>1</td>
<td>Jump if N = 1 or Z = 1</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JH</td>
<td>1</td>
<td>Jump if C = 1 and Z = 0</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JNH</td>
<td>1</td>
<td>Jump if C = 0 or Z = 1</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JV</td>
<td>1</td>
<td>Jump if V = 1</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JNV</td>
<td>1</td>
<td>Jump if V = 0</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JVT</td>
<td>1</td>
<td>Jump if VT = 1; Clear VT</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JNVT</td>
<td>1</td>
<td>Jump if VT = 0; Clear VT</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JST</td>
<td>1</td>
<td>Jump if ST = 1</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JNST</td>
<td>1</td>
<td>Jump if ST = 0</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>JBS</td>
<td>3</td>
<td>Jump if Specified Bit = 1</td>
<td>Z N C V VT ST</td>
<td>5,6</td>
</tr>
<tr>
<td>JBC</td>
<td>3</td>
<td>Jump if Specified Bit = 0</td>
<td>Z N C V VT ST</td>
<td>5,6</td>
</tr>
<tr>
<td>DJNZ</td>
<td>1</td>
<td>D ← D - 1; if D ≠ 0 then</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>DEC/DECB</td>
<td>1</td>
<td>D ← D - 1</td>
<td>Z N C V VT ST</td>
<td>5</td>
</tr>
<tr>
<td>NEG/NEG</td>
<td>1</td>
<td>D ← 0 - D</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>INC/INCB</td>
<td>1</td>
<td>D ← D + 1</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>EXT</td>
<td>1</td>
<td>D ← D; D + 2 ← Sign (D)</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>EXTB</td>
<td>1</td>
<td>D ← D; D + 1 ← Sign (D)</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>NOT/NOTB</td>
<td>1</td>
<td>D ← Logical Not (D)</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>CLR/CLRB</td>
<td>1</td>
<td>D ← 0</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>SHL/SHLB/SHLL</td>
<td>2</td>
<td>C ← msb ... lsb ← 0</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>SHR/SHRB/SHRL</td>
<td>2</td>
<td>0 → msb ... lsb → C</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>SHRA/SHRAB/SHRAL</td>
<td>2</td>
<td>msb → msb ... lsb → C</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>SETC</td>
<td>0</td>
<td>C ← 1</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>CLRC</td>
<td>0</td>
<td>C ← 0</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>CLRVT</td>
<td>0</td>
<td>VT ← 0</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>RST</td>
<td>0</td>
<td>PC ← 2080H</td>
<td>Z N C V VT ST</td>
<td>8</td>
</tr>
<tr>
<td>DI</td>
<td>0</td>
<td>Disable All Interrupts (I ← 0)</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>EI</td>
<td>0</td>
<td>Enable All Interrupts (I ← 1)</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>NOP</td>
<td>0</td>
<td>PC ← PC + 1</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>SKIP</td>
<td>0</td>
<td>PC ← PC + 2</td>
<td>Z N C V VT ST</td>
<td>-</td>
</tr>
<tr>
<td>NORM</td>
<td>2</td>
<td>Normalize (See sec 3.13.66)</td>
<td>Z N C V VT ST</td>
<td>7</td>
</tr>
<tr>
<td>TRAP</td>
<td>0</td>
<td>SP ← SP - 2; (SP) ← PC; PC ← (2010H)</td>
<td>Z N C V VT ST</td>
<td>9</td>
</tr>
</tbody>
</table>

Note
1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. Offset is a 2’s complement number.
3. Specified bit is one of the 2048 bits in the register file.
4. The “L” (Long) suffix indicates double-word operation.
5. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.
6. The assembler will not accept this mnemonic.

5-5
# Table 3-3. Opcode and State Time Listing

<table>
<thead>
<tr>
<th>MMEMONIC</th>
<th>OPERANDS</th>
<th>DIRECT</th>
<th>IMMEDIATE</th>
<th>INDIRECT</th>
<th>INDEXED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>OP CODE</td>
<td>BYTES</td>
<td>STATE TIMES</td>
<td>OP CODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>STATE TIMES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>2</td>
<td>64</td>
<td>3</td>
<td>4</td>
<td>65</td>
</tr>
<tr>
<td>ADD</td>
<td>3</td>
<td>44</td>
<td>4</td>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>ADDB</td>
<td>2</td>
<td>74</td>
<td>3</td>
<td>4</td>
<td>75</td>
</tr>
<tr>
<td>ADDB</td>
<td>3</td>
<td>54</td>
<td>4</td>
<td>5</td>
<td>55</td>
</tr>
<tr>
<td>ADDC</td>
<td>2</td>
<td>A4</td>
<td>3</td>
<td>4</td>
<td>A5</td>
</tr>
<tr>
<td>ADDCB</td>
<td>2</td>
<td>B4</td>
<td>3</td>
<td>4</td>
<td>B5</td>
</tr>
<tr>
<td>SUB</td>
<td>2</td>
<td>68</td>
<td>3</td>
<td>4</td>
<td>69</td>
</tr>
<tr>
<td>SUB</td>
<td>3</td>
<td>48</td>
<td>4</td>
<td>5</td>
<td>49</td>
</tr>
<tr>
<td>SUBB</td>
<td>2</td>
<td>78</td>
<td>3</td>
<td>4</td>
<td>79</td>
</tr>
<tr>
<td>SUBB</td>
<td>3</td>
<td>58</td>
<td>4</td>
<td>5</td>
<td>59</td>
</tr>
<tr>
<td>SUBC</td>
<td>2</td>
<td>A8</td>
<td>3</td>
<td>4</td>
<td>A9</td>
</tr>
<tr>
<td>SUBCB</td>
<td>2</td>
<td>B8</td>
<td>3</td>
<td>4</td>
<td>B9</td>
</tr>
<tr>
<td>CMP</td>
<td>2</td>
<td>88</td>
<td>3</td>
<td>4</td>
<td>89</td>
</tr>
<tr>
<td>CMPB</td>
<td>2</td>
<td>98</td>
<td>3</td>
<td>4</td>
<td>99</td>
</tr>
<tr>
<td>MUL</td>
<td>2</td>
<td>6C</td>
<td>3</td>
<td>25</td>
<td>6D</td>
</tr>
<tr>
<td>MUL</td>
<td>3</td>
<td>4C</td>
<td>4</td>
<td>26</td>
<td>4D</td>
</tr>
<tr>
<td>MUL</td>
<td>2</td>
<td>7C</td>
<td>3</td>
<td>17</td>
<td>7D</td>
</tr>
<tr>
<td>MUL</td>
<td>3</td>
<td>5C</td>
<td>4</td>
<td>18</td>
<td>5D</td>
</tr>
<tr>
<td>MUL</td>
<td>2</td>
<td>@</td>
<td>4</td>
<td>29</td>
<td>@</td>
</tr>
<tr>
<td>MUL</td>
<td>2</td>
<td>@</td>
<td>5</td>
<td>30</td>
<td>@</td>
</tr>
<tr>
<td>MULB</td>
<td>2</td>
<td>#</td>
<td>4</td>
<td>21</td>
<td>#</td>
</tr>
<tr>
<td>MULB</td>
<td>2</td>
<td>#</td>
<td>5</td>
<td>22</td>
<td>#</td>
</tr>
<tr>
<td>DIVU</td>
<td>2</td>
<td>8C</td>
<td>3</td>
<td>25</td>
<td>8D</td>
</tr>
<tr>
<td>DIVU</td>
<td>2</td>
<td>9C</td>
<td>3</td>
<td>17</td>
<td>9D</td>
</tr>
<tr>
<td>DIV</td>
<td>2</td>
<td>#</td>
<td>4</td>
<td>29</td>
<td>#</td>
</tr>
<tr>
<td>DIV</td>
<td>2</td>
<td>#</td>
<td>4</td>
<td>21</td>
<td>#</td>
</tr>
</tbody>
</table>

**Notes:**

- Long indexed and Indirect + instructions have identical opcodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.
- Number of state times shown for internal/external operands.
- The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.
Table 3-3. Continued

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPERANDS</th>
<th>DIRECT</th>
<th>IMMEDIATE</th>
<th>INDIRECT®</th>
<th>INDEXED®</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NORMAL</td>
<td>AUTO-INC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SHORT</td>
<td>LONG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STATE TIMES</td>
<td>STATE TIMES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OPCODE</td>
<td>BYTES</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STATE TIMES</td>
<td>STATE TIMES</td>
</tr>
</tbody>
</table>

**LOGICAL INSTRUCTIONS**

| AND   | 2  | 60  | 3  | 4  | 61  | 4  | 5  | 62  | 3  | 6/11 | 3  | 7/12 | 63  | 4  | 6/11 | 5  | 7/12 |
| ANDB  | 2  | 70  | 3  | 4  | 71  | 3  | 4  | 72  | 3  | 6/11 | 3  | 7/12 | 73  | 4  | 6/11 | 5  | 7/12 |
| ANDB  | 3  | 50  | 4  | 5  | 51  | 4  | 5  | 52  | 4  | 7/12 | 4  | 8/13 | 53  | 5  | 7/12 | 6  | 8/13 |
| OR    | 2  | 80  | 3  | 4  | 81  | 4  | 5  | 82  | 3  | 6/11 | 3  | 7/12 | 83  | 4  | 6/11 | 5  | 7/12 |
| ORB   | 2  | 90  | 3  | 4  | 91  | 3  | 4  | 92  | 3  | 6/11 | 3  | 7/12 | 93  | 4  | 6/11 | 5  | 7/12 |
| XOR   | 2  | 84  | 3  | 4  | 85  | 4  | 5  | 86  | 3  | 6/11 | 3  | 7/12 | 87  | 4  | 6/11 | 5  | 7/12 |
| XORB  | 2  | 94  | 3  | 4  | 95  | 3  | 4  | 96  | 3  | 6/11 | 3  | 7/12 | 97  | 4  | 6/11 | 5  | 7/12 |

**DATA TRANSFER INSTRUCTIONS**

| LD     | 2  | A0  | 3  | 4  | A1  | 4  | 5  | A2  | 3  | 6/11 | 3  | 7/12 | A3  | 4  | 6/11 | 5  | 7/12 |
| LDB    | 2  | B0  | 3  | 4  | B1  | 3  | 4  | B2  | 3  | 6/11 | 3  | 7/12 | B3  | 4  | 6/11 | 5  | 7/12 |
| ST     | 2  | C0  | 3  | 4  | —   | —   | —   | —   | C2  | 3  | 7/11 | 3  | 8/12 | C3  | 4  | 7/11 | 5  | 8/12 |
| STB    | 2  | C4  | 3  | 4  | —   | —   | —   | —   | C6  | 3  | 7/11 | 3  | 8/12 | C7  | 4  | 7/11 | 5  | 8/12 |
| LDBSE  | 2  | BC  | 3  | 4  | BD  | 3  | 4  | BE  | 3  | 6/11 | 3  | 7/12 | BF  | 4  | 6/11 | 5  | 7/12 |
| LDBZE  | 2  | AC  | 3  | 4  | AD  | 3  | 4  | AE  | 3  | 6/11 | 3  | 7/12 | AF  | 4  | 6/11 | 5  | 7/12 |

**STACK OPERATIONS (internal stack)**

| PUSH   | 1  | C8  | 2  | 8  | C9  | 3  | 8  | CA  | 2  | 11/15 | 2  | 12/16 | CB  | 3  | 11/15 | 4  | 12/16 |
| POP    | 1  | CC  | 2  | 12 | —   | —   | —   | —   | CE  | 2  | 14/18 | 2  | 14/18 | CF  | 3  | 14/18 | 4  | 14/18 |
| PUSHF  | 0  | F2  | 1  | 8  |      |      |      |      |      |      |      |      |      |      |      |      |      |
| POPF   | 0  | F3  | 1  | 9  |      |      |      |      |      |      |      |      |      |      |      |      |      |

**STACK OPERATIONS (external stack)**

| PUSH   | 1  | C8  | 2  | 12 | C9  | 3  | 12 | CA  | 2  | 15/19 | 2  | 16/20 | CB  | 3  | 15/19 | 4  | 16/20 |
| POP    | 1  | CC  | 2  | 14 | —   | —   | —   | —   | CE  | 2  | 16/20 | 2  | 16/20 | CF  | 3  | 16/20 | 4  | 16/20 |
| PUSHF  | 0  | F2  | 1  | 12 |      |      |      |      |      |      |      |      |      |      |      |      |      |
| POPF   | 0  | F3  | 1  | 13 |      |      |      |      |      |      |      |      |      |      |      |      |      |

**JUMPS AND CALLS**

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>Bytes</th>
<th>States</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>Bytes</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>LJMP</td>
<td>E7</td>
<td>3</td>
<td>8</td>
<td>LCALL</td>
<td>EF</td>
<td>3</td>
<td>13/16()</td>
</tr>
<tr>
<td>SJMP</td>
<td>20-27()</td>
<td>2</td>
<td>8</td>
<td>SCALL</td>
<td>28-2F()</td>
<td>2</td>
<td>13/16()</td>
</tr>
<tr>
<td>BR[ ]</td>
<td>E3</td>
<td>2</td>
<td>8</td>
<td>RET</td>
<td>F0</td>
<td>1</td>
<td>12/16()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TRAP()</td>
<td>F7</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- Number of state times shown for internal/external operands.
- The assembler does not accept this mnemonic.
- The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.
- State times for stack located internal/external.
- The assembler uses the generic jump mnemonic (BR) to generate this instruction.
### Table 3-4. CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>JC</td>
<td>DB</td>
<td>JE</td>
<td>DF</td>
<td>JGE</td>
<td>D6</td>
<td>JGT</td>
<td>D2</td>
</tr>
<tr>
<td>JNC</td>
<td>D3</td>
<td>JNE</td>
<td>D7</td>
<td>JLT</td>
<td>DE</td>
<td>JLE</td>
<td>DA</td>
</tr>
<tr>
<td>JH</td>
<td>D9</td>
<td>JV</td>
<td>DD</td>
<td>JVT</td>
<td>DC</td>
<td>JST</td>
<td>D8</td>
</tr>
<tr>
<td>JNH</td>
<td>D1</td>
<td>JNV</td>
<td>D5</td>
<td>JNVT</td>
<td>D4</td>
<td>JNST</td>
<td>D0</td>
</tr>
</tbody>
</table>

### JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

<table>
<thead>
<tr>
<th>BIT NUMBER</th>
<th>MNEMONIC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JBC</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>JBS</td>
<td>38</td>
<td>39</td>
<td>3A</td>
<td>3B</td>
<td>3C</td>
<td>3D</td>
<td>3E</td>
<td>3F</td>
</tr>
</tbody>
</table>

### LOOP CONTROL

DINZ OPCODE EO; 3 BYTES; 5/9 STATE TIMES (NOT TAKEN/TAKEN)

### SINGLE REGISTER INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>05</td>
<td>2</td>
<td>4</td>
<td>EXT</td>
<td>06</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>DECB</td>
<td>15</td>
<td>2</td>
<td>4</td>
<td>EXTB</td>
<td>16</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>NEG</td>
<td>03</td>
<td>2</td>
<td>4</td>
<td>NOT</td>
<td>02</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>NEGB</td>
<td>13</td>
<td>2</td>
<td>4</td>
<td>NOTB</td>
<td>12</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>INC</td>
<td>07</td>
<td>2</td>
<td>4</td>
<td>CLR</td>
<td>01</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>INCB</td>
<td>17</td>
<td>2</td>
<td>4</td>
<td>CLRB</td>
<td>11</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

### SHIFT INSTRUCTIONS

<table>
<thead>
<tr>
<th>INSTR WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>INSTR</th>
<th>WORD</th>
<th>STATE TIMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHL</td>
<td>09</td>
<td>3</td>
<td>SHLB</td>
<td>19</td>
<td>3</td>
<td>SHLL</td>
<td>0D</td>
<td>3</td>
<td>7 + 1 PER SHIFT©</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHR</td>
<td>08</td>
<td>3</td>
<td>SHRAB</td>
<td>1A</td>
<td>3</td>
<td>SHRAL</td>
<td>0E</td>
<td>3</td>
<td>7 + 1 PER SHIFT©</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHRA</td>
<td>0A</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SPECIAL CONTROL INSTRUCTIONS

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETC</td>
<td>F9</td>
<td>1</td>
<td>4</td>
<td>DI</td>
<td>FA</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>CLRC</td>
<td>F8</td>
<td>1</td>
<td>4</td>
<td>EI</td>
<td>FB</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>CLRVT</td>
<td>FC</td>
<td>1</td>
<td>4</td>
<td>NOP</td>
<td>FD</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>RST</td>
<td>FF</td>
<td>1</td>
<td>16</td>
<td>SKIP</td>
<td>00</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

### NORMALIZE

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>OPCODE</th>
<th>BYTES</th>
<th>STATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORML</td>
<td>0F</td>
<td>3</td>
<td>11 + 1 PER SHIFT</td>
</tr>
</tbody>
</table>

**Notes:**

© This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.

© Execution will take at least 8 states, even for 0 shift.
ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias ........... -0°C to +70°C
Storage Temperature ................... -40°C to +150°C
Voltage from Any Pin to VSS or ANGND .... -0.3 V to +7 V
Average Output Current from Any Pin ....... 10 mA
Power Dissipation .................. 1.5 Watts

"NOTICE Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

OPERATING CONDITIONS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>Ambient Temperature Under Bias</td>
<td>0°C</td>
<td>+70°C</td>
<td>C</td>
</tr>
<tr>
<td>VCC</td>
<td>Digital Supply Voltage</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>VREF</td>
<td>Analog Supply Voltage</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
</tr>
<tr>
<td>fOSC</td>
<td>Oscillator Frequency</td>
<td>6.0</td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td>VPD</td>
<td>Power-Down Supply Voltage</td>
<td>4.5</td>
<td>5.5</td>
<td>V</td>
</tr>
</tbody>
</table>

NOTICE: VBB should be connected to ANGND through a 0.01 μF capacitor. ANGND and VSS should be nominally at the same potential.

DC CHARACTERISTICS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.3</td>
<td>+0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (Except RESET)</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage, RESET Rising</td>
<td>2.4</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH2</td>
<td>Input High Voltage, RESET Falling</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage</td>
<td>0.45</td>
<td>V</td>
<td>See Note 1.</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td></td>
<td>See Note 2.</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>VCC Supply Current</td>
<td>200</td>
<td>mA</td>
<td>All outputs disconnected.</td>
<td></td>
</tr>
<tr>
<td>IPD</td>
<td>VPD Supply Current</td>
<td>1</td>
<td>mA</td>
<td>Normal operation and Power-Down.</td>
<td></td>
</tr>
<tr>
<td>IREF</td>
<td>VREF Supply Current</td>
<td>15</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IL1</td>
<td>Input Leakage Current to all pins of HSI, P0, P3, P4, and to P2.1.</td>
<td>±10</td>
<td>μA</td>
<td>Vin = 0 to VCC See Note 3</td>
<td></td>
</tr>
<tr>
<td>IIH</td>
<td>Input High Current to EA</td>
<td>100</td>
<td>μA</td>
<td>VIH = 2.4V</td>
<td></td>
</tr>
<tr>
<td>II2</td>
<td>Input Low Current to all pins of P1, and to P2.6, P2.7.</td>
<td>-100</td>
<td>μA</td>
<td>VIL = 0.45V</td>
<td></td>
</tr>
<tr>
<td>III1</td>
<td>Input Low Current to RESET</td>
<td>-2</td>
<td>mA</td>
<td>VIL = 0.45V</td>
<td></td>
</tr>
<tr>
<td>III2</td>
<td>Input Low Current P2.2, P2.3, P2.4, READY</td>
<td>-50</td>
<td>μA</td>
<td>VIL = 0.45V</td>
<td></td>
</tr>
<tr>
<td>Cs</td>
<td>Pin Capacitance (Any Pin to VSS)</td>
<td>10</td>
<td>pF</td>
<td>fTEST = 1MHz</td>
<td></td>
</tr>
</tbody>
</table>

NOTES:

1. IOL = 0.36 mA for all pins of P1, for P2 6 and P2 7, and for all pins of P3 and P4 when used as ports.
2. IOL = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, RD, WR, and all pins of HSO and P3 and P4 when used as external memory bus (ADD-AD15).
3. IOH = -0.20 μA for all pins of P1, for P2 6 and P2 7.
4. IOH = -0.200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, BHE, WR, and all pins of HSO and P3 and P4 when used as external memory bus (ADD-AD15).
5. P3 and P4, when used as ports, have open-drain outputs.
6. Analog Conversion not in process.
A/D CONVERTER SPECIFICATIONS
A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120 volts.

AC CHARACTERISTICS
Test Conditions: Load capacitance on output pins = 80pf
Oscillator Frequency = 12.00 MHz
4.50 Volts , VCC , = 5.50 Volts; 0 C , Temperature , = 70 C

Timing Requirements (other system components must meet these specs)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLYX</td>
<td>READY Hold after CLKOUT falling edge</td>
<td>0 (1)</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TLLYV</td>
<td>End of ALE to READY Setup</td>
<td></td>
<td>2Tosc - 60</td>
<td>nsec</td>
</tr>
<tr>
<td>TLLYH</td>
<td>End of ALE to READY high</td>
<td></td>
<td>4Tosc - 60 (2)</td>
<td>nsec</td>
</tr>
<tr>
<td>TYLYH</td>
<td>Non-ready time</td>
<td></td>
<td>1000</td>
<td>nsec</td>
</tr>
<tr>
<td>TAVDV</td>
<td>Address Valid to Input Data Valid</td>
<td></td>
<td>5Tosc - 80</td>
<td>nsec</td>
</tr>
<tr>
<td>TRLDV</td>
<td>RD/ Active to Input Data Valid</td>
<td></td>
<td>3Tosc - 60</td>
<td>nsec</td>
</tr>
<tr>
<td>TRXHZ</td>
<td>End of RD/ to Input Data Float</td>
<td></td>
<td>Tosc - 20</td>
<td>nsec</td>
</tr>
</tbody>
</table>

Timing Responses (MCS®-96 parts meet these specs)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>FXTAL</td>
<td>Oscillator Frequency</td>
<td>6.00</td>
<td>12.00</td>
<td>MHz</td>
</tr>
<tr>
<td>Tosc</td>
<td>Oscillator Period</td>
<td>83</td>
<td>166</td>
<td>nsec</td>
</tr>
<tr>
<td>TCHCH</td>
<td>CLKOUT Period</td>
<td>3Tosc (3)</td>
<td>3Tosc (3)</td>
<td>nsec</td>
</tr>
<tr>
<td>TCHCL</td>
<td>CLKOUT High Time</td>
<td>Tosc - 20</td>
<td>Tosc + 20</td>
<td>nsec</td>
</tr>
<tr>
<td>TCCLH</td>
<td>CLKOUT Low to ALE High</td>
<td>-5</td>
<td>20</td>
<td>nsec</td>
</tr>
<tr>
<td>TLLCH</td>
<td>ALE Low to CLKOUT High</td>
<td>Tosc - 20</td>
<td>Tosc + 40</td>
<td>nsec</td>
</tr>
<tr>
<td>TLHLL</td>
<td>ALE Pulse Width</td>
<td>Tosc - 25</td>
<td>Tosc + 15</td>
<td>nsec</td>
</tr>
<tr>
<td>TAVLL</td>
<td>Address Setup to End of ALE</td>
<td>Tosc - 50</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TLLRL</td>
<td>End of ALE to RD/ or WR/ active</td>
<td>Tosc - 20</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address hold after End of ALE</td>
<td>Tosc - 20</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR/ Pulse Width</td>
<td>2Tosc - 35</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TQVWX</td>
<td>Output Data Setup to End of WR/</td>
<td>2Tosc - 60</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TWXQX</td>
<td>Output Data Hold after End of WR/</td>
<td>Tosc - 25</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TWXLH</td>
<td>End of WR/ to next ALE</td>
<td>2Tosc - 30</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TRLRH</td>
<td>RD/ Pulse Width</td>
<td>3Tosc - 30</td>
<td></td>
<td>nsec</td>
</tr>
<tr>
<td>TRHLH</td>
<td>End of RD/ to next ALE</td>
<td>Tosc - 25</td>
<td></td>
<td>nsec</td>
</tr>
</tbody>
</table>

NOTES:
1. If the 48-pin part is being used then this timing can be generated by assuming that the CLKOUT falling edge has occurred at 2Tosc + 55 (TLLCH(max) + TCHCL(max)) after the falling edge of ALE.
2. If more than one wait state is desired, add 3Tosc for each additional wait state.
3. CLKOUT is directly generated as a divide by 3 of the oscillator. The period will be 3Tosc ± 5 nsec if Tosc is constant and the rise and fall times on XTAL 1 are less than 10 nsec.
High-Performance Event Interface for a Microcomputer
As silicon technology advances to provide denser geometries, timer structures have become more elegant and powerful.

Microcontrollers are microprocessors specially configured to monitor and control mechanisms and processes rather than manipulate data. The systems they are embedded in are often called real time control systems; microcontrollers always incorporate some form of timer structure to allow synchronization with the outside or 'real' world. As silicon technology advances to provide denser geometries, these structures have become more elegant and powerful.

This trend can be seen in the Intel 8048, the Motorola 6801, and the Intel 8051 which were introduced at approximately two and a half year intervals starting in 1976. The 8048 has a single 8-bit timer; the 6801 has a 16-bit timer, and the 8051 has two 16-bit timers. The new 16-bit microcontroller from Intel, the 8096, has an independent High Speed 1/O subsystem which provides the functionality of four to eight 16-bit timers. While this subsystem is designed to provide an integrated approach to measuring and controlling time modulated signals, it is easier to describe as separate input and output units.

**High Speed Input Unit**

The purpose of the High Speed Input unit is to allow the measurement of the periods of incoming pulse or frequency modulated inputs with high resolution and minimal software overhead. A block diagram of the hardware used to accomplish this goal is shown in Figure 1. The heart of this unit is a programmable change detector which monitors the four 1/O pins of the 8096 which are designated as "High Speed Inputs" (HSI.0-HSI.3).

The operating mode of the change detector is controlled by a byte register which can be written as register 3 of the onboard register file. This register has the predeclared name HSI_MODE in the 8096 assembly language. The register contains a separate field for each of the four HSI pins. There are two bits in each of these fields and they are encoded as follows:

- 00 Capture every eighth positive transition
- 01 Capture positive transitions only
- 10 Capture negative transitions only
- 11 Capture both positive and negative transitions

It is also possible to disconnect one or more of the HSI pins from the change detector by writing into one of the two I/O control registers. This register, known to the assembly language as I0C0 is addressed as register 15H of the on-board register file. HSI pins that

---

**Figure 1:** Diagram shows the High-Speed Input Unit which is used to measure incoming pulse or frequency modulated inputs.
have to be disconnected from the change detector are available for use as normal digital inputs and two of them (HSI.2 and HSI.3) can be used by the High Speed Output unit.

When a change (or changes) of the required type occurs, four bits of change information, along with the current value of TIMER1, are loaded into a FIFO (first-in, first-out memory). Each set bit in this field indicates that a change occurred on the corresponding input pin.

The time reference for the HSI unit is TIMER1, a sixteen bit counter which is incremented every eight state times by the CPU clock. With a 12 MHz crystal this gives a resolution of 2.0 microseconds. TIMER1 is cleared by reset and then starts incrementing. It cannot be written to by the software but can be read as a sixteen bit word at any time. When its count goes from all ones to zero a flag is set and an interrupt generated. The software can use this flag and/or interrupt to extend the measurement range of the HSI unit.

The FIFO that is used to store the change and time information is eight levels deep (including the holding register) and 20 bits wide. The oldest entry in the FIFO is placed in the holding register. When the holding register is read then the next oldest entry will drop into it and another cell of the FIFO will become available for input data. An interrupt can be generated either when one or more entries exist in the FIFO or when seven or more entries exist. The choice is made by the software by setting a bit in I/O control register 1 (IOC1).

The 8096 only supports byte and word operands for most operations. The holding register is 20 bits wide hence the holding register is broken down into two registers. The 16-bit time field is read as a word register and is known as HSL_TIME to the assembler. The change information is read as an eight-bit byte known as HSL_STATUS.

The four extra bits in this byte are used to report the state of the HSI pins at the time the register is read (not at the time the reported change occurred). The holding register is cleared after the HSL_TIME is read so that HSL_STATUS can be read at any time to monitor the actual state of the HSI pins without losing data from the FIFO.

High Speed Output Unit

The High Speed Output unit serves the output requirements of the system in the same way as the HSI unit serves the input. It allows the generation of pulse and frequency modulated signals with high resolution and minimal software overhead. It can also be used to generate time delays for the operating software and to trigger the A/D converter at precise time intervals for signal processing algorithms. A block diagram of the HSO hardware is shown in Figure 2.

The HSO unit is driven by a Content Addressable Memory (CAM) which is 23 bits wide and eight levels deep. The

(continued on page 120)
23 bits are broken into a 16 bit time tag and a seven bit command tag. The command tag tells it when to do it. The format of the command tag is shown in Figure 3. The lower four bits of the tag specify the basic operation and the remaining three bits specify options to the basic operation. The basic operations supported are:

- Write to one of the six pins controlled by the HSO unit (HSO.0-HSO.5).
- Write to HSO.0 and HSO.1 with a single command.
- Write to HSO.2 and HSO.3 with a single command.
- Set one of four software timer flags.
- Reset Timer 2.
- Trigger an A/D conversion.

If an operation on an HSO pin is specified, then the value to be written to the pin is taken from bit five of the command tag. Note that if two HSO pins are to be modified with the same command then both will be set to the same state. Bit five of the command tag is ignored for the other HSO operations. Bit four of the command tag enables the generation of an interrupt which occurs when the command is executed.

There are two interrupts generated by the HSO unit. One of them indicates that an operation involving a HSO pin has occurred, and the other is used to signal that one of the internal HSO functions (such as setting a software timer flag) has been completed. Bit six of the command tag controls which one of the two timers available to the HSO unit will be used as a time base for the command. If bit six is a zero then the command tag will be executed when TIMER1 becomes equal to the time tag stored in the CAM. If bit six is a one then the command tag is executed based on TIMER2. In either case the command is flushed from the CAM as soon as it is executed.

One of the timers (TIMER1) used by the HSO unit is the same timer that is used by the HSI unit. The other (TIMER2) is used only by the HSO unit. TIMER2 allows HSO events to be generated on a time base that is different from that of the CPU. Like TIMER1 it is a 16 bit counter that can be read but not written to by the software. It also has an overflow flag and interrupt to indicate that it has incremented from a source external to the 8096 and can be reset by a number of paths in addition to system reset. The options available are shown in Figure 4.

The clock input can come either from a specific pin designated as the T2CLK or it can come from HSI.1 depending on the state of IOCO.7 which is set by the software. In either case the counter is incremented on both edges of the clock signal. TIMER2 can be reset by a specific pin designated as T2RST or it can be reset by HSO.0. It is also possible for the software to lock out external sources of reset (by clearing IOCO.3) and/or reset TIMER2 directly (via IOCO.1) or indirectly via a command stored in the CAM. Note that this last possibility allows TIMER2 to be configured as a modulo counter since the software can command the HSO unit to clear TIMER2 when it reaches a given value.

Commands are loaded into the CAM from the 23 bit wide holding register which, like the holding register for the HSI unit, is actually made up of a byte register (HSO_COMMAND) which stores the command tag and a word register is considered loaded after HSO_TIME is loaded so the software must always load HSO_COMMAND and then load HSO_TIME.

The software must also ensure that the loading of the two registers is not interrupted by an interrupt service routine which uses the HSO unit. If such an interrupt occurs immediately following the loading of HSO_COMMAND then the subsequent loading of HSO_TIME will reload the command tag written into the holding register by the interrupt service routine. The safest procedure is to lock out interrupts during the loading of the holding registers, however a careful examination of the control flow of the program may show this to be unnecessary.

If there is an empty cell in the CAM when the holding register is loaded then the command and its time tag will be loaded into the CAM within seven state times (1.75 microseconds at 12 MHz). It is important to note that a command will not execute from the holding register, it must be loaded into the CAM. If the CAM is full then the command will remain in the holding register until one of the commands already in the CAM is executed and flushed.

Two status flags are available to help the software manage the CAM. One of them indicates that the holding register is full or the CAM is full. Once a command is loaded into the CAM it cannot be read or overwritten, it can only be flushed after it is executed. To support those situations where the software wishes to cancel a command after it has been loaded, the HSO unit is configured so that two operations to a HSO pin which cancel each other will not effect the setting of the pin if they are executed with identical time tags.

**Application Example**

Since the 8096 incorporates a full duplex asynchronous serial port in its hardware it may seem strange that one would want to implement a software driven serial port using the high speed I/O features. There are, however, many useful configurations of microcontroller systems which in fact require more than a single serial port. An obvious example would be a network of 8096 controllers which use the hardware serial port for interprocessor communications. One (or more) of these controllers might also be required to communicate with a CRT terminal used to supervise or monitor

---

**Figure 5:** Figure shows standard 10-bit asynchronous frame.
The serial output process is simpler than the receive process because there is no need to synchronize with the outside world.

the system. Another example would be a simple CRT terminal design based on an 8096 which needs one serial port for communication and another for driving a slave printer. It may also be true that this is, in fact, a strange requirement. In any case it is an excellent example to show how the high speed I/O features of the 8096 might be used.

The objective is to add a software driven asynchronous serial port to the 8096 that provides full duplex serial communication at 2400 baud. A standard frame consisting of a START bit, eight data bits and a STOP bit will be assumed. A high speed input pin (HSI.0) will be used for received data and a high speed output pin (HSI.0) for transmit data.

A standard 10-bit asynchronous frame is shown in Figure 5. The figure also shows the points in time where the receive process must sample the incoming data stream and take some action. The first timing point (labeled T=0) is the leading edge of the start bit, the accurate sensing of this edge is important because all subsequent sample times are relative to this edge. This event also places the highest burden on the sampling algorithm because it can occur at any point in time. The rest of the sampling events occur at some multiple of one-half a bit period relative to the edge of the start bit. The diagram uses the symbol B to represent a bit period.

At the second sample, which occurs half way through the start bit, the data must be checked to make sure it is still a SPACE. If it is not, a noise pulse has caused a false start and the receive process must be reinitialized. The next eight samples are used to shift in the serial data stream. The last sample, which occurs 19 one-half bit times after the leading edge of the start bit, is used to verify that the stop bit is valid (i.e. it is in the MARK state). If it is not the a framing error must be reported since it is likely that the receiver is not properly synchronized with the transmitter.

The HSI unit is an ideal mechanism for detecting the leading edge of the start bit. All that needs to be done is to set the mode register to detect negative going edges on HSI.0.

The software timer interrupt service routine implements a simple state machine based on the variable count. The routine also arranges for the next sample by issuing a command to the HSO unit to generate another software timer interrupt at the appropriate time. This is done in all states unless the reception of the character is complete or a false START bit has been detected. Under these conditions the receive process must be reinitialized by enabling HSI.0 into the event FIFO (by setting IOC.0) instead of retriggering the software timer.

The serial output process is simpler than the receive process because there is no need to synchronize with the outside world. A transmission can be started at any time by setting the TxD line to a space for one bit time to form the START bit. Following the START bit are the eight data bits and the STOP bit. The HSO interrupt service routine can be used to transmit the data and the stop bit but the transmit process must be initialized.

The only real complication in the HSO interrupt service routine is that there are no flags available in the 8096 which indicate which of the HSO outputs caused the interrupt. In many systems this does not represent a problem because the HSO unit can be treated as a write only device. It is given commands which are to be executed at the proper time but no feedback is required to indicate when the proper time has been reached. In this case, however, the feedback is required since the CAM isn't deep enough to hold all of the transitions required for a character. Even if it were big enough it is unlikely that so many CAM locations would be dedicated to serial output.

6-5
7.0 INTRODUCTION

The MCS®-51 family of 8-bit microcontrollers consists of the devices listed in Table 1, all of which are based on the MCS-51 architecture shown in Figure 7-1. The original 8051 was built in HMOS I technology. The HMOS II version, which is the device currently in production, is called the 8051AH. The term "8051," however, is still often used to generically refer to all of the MCS-51 family members. This is the case throughout this manual, except where specifically stated otherwise. Also for brevity, the term "8052" is used to refer to both the 8052 and the 8032, unless otherwise noted.

Figure 7-1. MCS-51 Architectural Block Diagram
The newest MCS-51 members, the 8032 and 8052, have more on-chip memory and an additional 16-bit timer/counter. The new timer can be used as a timer, a counter, or to generate baud rates for the serial port. As a timer/counter, it operates in either a 16-bit auto-reload mode or a 16-bit "capture" mode. This new feature is described in Section 7.6.2.

Pinouts are shown in the individual data sheets and on the inside back cover of this handbook.

Table 1. MCS®-51 Family Members

<table>
<thead>
<tr>
<th>PART</th>
<th>TECHNOLOGY</th>
<th>ON-CHIP PROGRAM MEMORY</th>
<th>ON-CHIP DATA MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>8051</td>
<td>HMOS</td>
<td>4K — ROM</td>
<td>128</td>
</tr>
<tr>
<td>8031</td>
<td>HMOS</td>
<td>NONE</td>
<td>128</td>
</tr>
<tr>
<td>8751H</td>
<td>HMOS I</td>
<td>4K — EPROM</td>
<td>128</td>
</tr>
<tr>
<td>80C51</td>
<td>CHMOS</td>
<td>4K — ROM</td>
<td>128</td>
</tr>
<tr>
<td>80C31</td>
<td>CHMOS</td>
<td>NONE</td>
<td>128</td>
</tr>
<tr>
<td>8052</td>
<td>HMOS</td>
<td>8K — ROM</td>
<td>256</td>
</tr>
<tr>
<td>8032</td>
<td>HMOS</td>
<td>NONE</td>
<td>256</td>
</tr>
</tbody>
</table>

The major MCS®-51 features are:

- 8-Bit CPU
- On-Chip oscillator and clock circuitry
- 32 I/O lines
- 64K address space for external data memory
- 64K address space for external program memory
- Two 16-bit timer/counters (three on 8032/8052)
- A five-source interrupt structure (six sources on 8032/8052) with two priority levels
- Full duplex serial port
- Boolean processor

7.1 MEMORY ORGANIZATION

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K (8K for 8052) may reside on-chip. The Data Memory can consist of up to 64K bytes of off-chip RAM, in addition to which it includes 128 bytes of on-chip RAM (256 bytes for the 8052), plus a number of "SFRs" (Special Function Registers) as listed below.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>*P2</td>
<td>Port 2</td>
<td>0A0H</td>
</tr>
<tr>
<td>*P3</td>
<td>Port 3</td>
<td>0B0H</td>
</tr>
<tr>
<td>*IP</td>
<td>Interrupt Priority Control</td>
<td>0B8H</td>
</tr>
<tr>
<td>*IE</td>
<td>Interrupt Enable Control</td>
<td>0A8H</td>
</tr>
<tr>
<td>TMOD</td>
<td>Timer/Counter Mode Control</td>
<td>89H</td>
</tr>
<tr>
<td>*TCON</td>
<td>Timer/Counter Control</td>
<td>88H</td>
</tr>
<tr>
<td>+*T2CON</td>
<td>Timer/Counter 2 Control</td>
<td>0C8H</td>
</tr>
<tr>
<td>TH0</td>
<td>Timer/Counter 0 (high byte)</td>
<td>0CH</td>
</tr>
<tr>
<td>TL0</td>
<td>Timer/Counter 0 (low byte)</td>
<td>0AH</td>
</tr>
<tr>
<td>TH1</td>
<td>Timer/Counter 1 (high byte)</td>
<td>0DH</td>
</tr>
<tr>
<td>TL1</td>
<td>Timer/Counter 1 (low byte)</td>
<td>0BH</td>
</tr>
<tr>
<td>+TH2</td>
<td>Timer/Counter 2 (high byte)</td>
<td>0CDH</td>
</tr>
<tr>
<td>+TL2</td>
<td>Timer/Counter 2 (low byte)</td>
<td>0CCH</td>
</tr>
<tr>
<td>+RCAP2H</td>
<td>Timer/Counter 2 Capture Register (high byte)</td>
<td>0CBH</td>
</tr>
<tr>
<td>+RCAP2L</td>
<td>Timer/Counter 2 Capture Register (low byte)</td>
<td>0CAH</td>
</tr>
<tr>
<td>*SCON</td>
<td>Serial Control</td>
<td>98H</td>
</tr>
<tr>
<td>SBUF</td>
<td>Serial Data Buff</td>
<td>99H</td>
</tr>
<tr>
<td>PCON</td>
<td>Power Control</td>
<td>87H</td>
</tr>
</tbody>
</table>

The SFRs marked with an asterisk (*) are both bit- and byte-addressable. The SFRs marked with a plus sign (+) are present in the 8052 only. The functions of the SFRs are described as follows.

ACCUMULATOR

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

B REGISTER

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

PROGRAM STATUS WORD

The PSW register contains program status information as detailed in Figure 7-2.

STACK POINTER

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM,
the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

**DATA POINTER**

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

**PORTS 0 to 3**

P0, P1, P2 and P3 are the SFR latches of Ports 0, 1, 2 and 3, respectively.

**SERIAL DATA BUFFER**

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

**TIMER REGISTERS**

Register pairs (TH0, TL0), (TH1, TL1), and (TH2, TL2) are the 16-bit counting registers for Timer/Counters 0, 1, and 2, respectively.

**CAPTURE REGISTERS**

The register pair (RCAP2H, RCAP2L) are the capture registers for the Timer 2 "capture mode." In this mode, in response to a transition at the 8052’s T2EX pin, TH2 and TL2 are copied into RCAP2H and RCAP2L. Timer 2 also has a 16-bit auto-reload mode, and RCAP2H and RCAP2L hold the reload value for this mode. More about Timer 2’s features in Section 7.6.2.

**CONTROL REGISTERS**

Special Function Registers IP, IE, TMOD, TCON, T2CON, SCON, and PCON contain control and status bits for the interrupt system, the timer/counters, and the serial port. They are described in later sections.

### 7.2 OSCILLATOR AND CLOCK CIRCUIT

XTAL1 and XTAL2 are the input and output of a single-stage on-chip inverter, which can be configured with off-chip components as a Pierce oscillator, as shown in Figure 7-3. The on-chip circuitry, and selection of off-chip components to configure the oscillator are discussed in Section

![Figure 7-3: Crystal/Ceramic Resonator Oscillator](image)

#### Table: PSW: Program Status Word Register

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CY</td>
<td>PSW.7</td>
<td>Carry flag.</td>
<td>OV</td>
<td>PSW.2</td>
<td>Overflow flag.</td>
</tr>
<tr>
<td>AC</td>
<td>PSW.6</td>
<td>Auxiliary Carry flag. (For BCD operations.)</td>
<td>—</td>
<td>PSW.1</td>
<td>(reserved)</td>
</tr>
<tr>
<td>F0</td>
<td>PSW.5</td>
<td>Flag 0 (Available to the user for general purposes.)</td>
<td>P</td>
<td>PSW.0</td>
<td>Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of &quot;one&quot; bits in the accumulator, i.e., even parity.</td>
</tr>
<tr>
<td>RS1</td>
<td>PSW.4</td>
<td>Register bank Select control bits 1 &amp; 0. Set/cleared by software to determine working register bank (see Note).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RS0</td>
<td>PSW.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note— the contents of (RS1, RS0) enable the working register banks as follows:

- Bank 0: 00H-07H
- Bank 1: 08H-0FH
- Bank 2: 10H-17H
- Bank 3: 18H-1FH

![Figure 7-2: PSW: Program Status Word Register](image)
7.13. A more detailed discussion will be found in Application Note AP-155, “Oscillators for Microcontrollers,” which is included in this manual.

The oscillator, in any case, drives the internal clock generator. The clock generator provides the internal clocking signals to the chip. The internal clocking signals are at half the oscillator frequency, and define the internal phases, states, and machine cycles, which are described in the next section.

7.3 CPU TIMING

A machine cycle consists of 6 states (12 oscillator periods). Each state is divided into a Phase 1 half, during which the Phase 1 clock is active, and a Phase 2 half.

![Diagram showing CPU timing phases](image-url)
during which the Phase 2 clock is active. Thus, a machine cycle consists of 12 oscillator periods, numbered S1P1 (State 1, Phase 1), through S6P2 (State 6, Phase 2). Each phase lasts for one oscillator period. Each state lasts for two oscillator periods. Typically, arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place during Phase 2.

The diagrams in Figure 7-4 show the fetch/execute timing referenced to the internal states and phases. Since these internal clock signals are not user accessible, the XTAL2 oscillator signal and the ALE (Address Latch Enable) signal are shown for external reference. ALE is normally activated twice during each machine cycle: once during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the Instruction Register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one-byte instruction, there is still a fetch at S4, but the byte read (which would be the next opcode), is ignored, and the Program Counter is not incremented. In any case, execution is complete at the end of S6P2. Figures 7-4A and 7-4B show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most 8051 instructions execute in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete. They take four cycles.

Normally, two code bytes are fetched from Program Memory during every machine cycle. The only exception to this is when a MOVX instruction is executed. MOVX is a 1-byte 2-cycle instruction that accesses external Data Memory. During a MOVX, two fetches are skipped while the external Data Memory is being addressed and strobed. Figures 7-4C and 7-4D show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

### 7.4 PORT STRUCTURES AND OPERATION

All four ports in the 8051 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

---

**Figure 7-5. 8051 Port Bit Latches and I/O Buffers**

*See Figure 7-6 for details of the internal pullup.*
The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external memory address when the address is 16 bits wide. Otherwise the Port 2 pins continue to emit the P2 SFR content.

All the Port 3 pins, and (in the 8052) two Port 1 pins are multifunctional. They are not only port pins, but also serve the functions of various special features as listed below:

<table>
<thead>
<tr>
<th>PORT PIN</th>
<th>ALTERNATE FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>*P1.0</td>
<td>T2 (Timer/Counter 2 external input)</td>
</tr>
<tr>
<td>*P1.1</td>
<td>T2EX (Timer/Counter 2 capture/reload trigger)</td>
</tr>
<tr>
<td>P3.0</td>
<td>RXD (serial input port)</td>
</tr>
<tr>
<td>P3.1</td>
<td>TXD (serial output port)</td>
</tr>
<tr>
<td>P3.2</td>
<td>INT0 (external interrupt)</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 (external interrupt)</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 (Timer/Counter 0 external input)</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 (Timer/Counter 1 external input)</td>
</tr>
<tr>
<td>P3.6</td>
<td>WR (external Data memory write strobe)</td>
</tr>
<tr>
<td>P3.7</td>
<td>RD (external Data memory read strobe)</td>
</tr>
</tbody>
</table>

*P1.0 and P1.1 serve these alternate functions only on the 8052.

The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise the port pin is stuck at 0.

### 7.4.1 I/O Configurations

Figure 7-5 shows a functional diagram of a typical bit latch and I/O buffer in each of the four ports. The bit latch (one bit in the port's SFR) is represented as a Type D flip-flop, which will clock in a value from the internal bus in response to a "write to latch" signal from the CPU. The Q output of the flip-flop is placed on the internal bus in response to a "read latch" signal from the CPU. The level of the port pin itself is placed on the internal bus in response to a "read pin" signal from the CPU. Some instructions that read a port activate the "read latch" signal, and others activate the "read pin" signal. More about that in Section 7.4.4.

As shown in Figure 7-5, the output drivers of Ports 0 and 2 are switchable to an internal ADDR and ADDR/DATA bus by an internal CONTROL signal for use in external memory accesses. During external memory accesses, the P2 SFR remains unchanged, but the P0 SFR gets 1s written to it.

Also shown in Figure 7-5, is that if a P3 bit latch contains a 1, then the output level is controlled by the signal labeled "alternate output function." The actual P3.X pin level is always available to the pin's alternate input function, if any.

Ports 1, 2, and 3 have internal pull-ups. Port 0 has open-drain outputs. Each I/O line can be independently used as an input or an output. (Ports 0 and 2 may not be used as general purpose I/O when being used as the ADDR/DATA BUS). To be used as an input, the port bit latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pull-up, but can be pulled low by an external source.

Port 0 differs in not having internal pullups. The pullup FET in the P0 output driver (see Figure 7-5A) is used only when the Port is emitting 1s during external memory accesses. Otherwise the pullup FET is off. Consequently P0 lines that are being used as output port lines are open drain. Writing a 1 to the bit latch leaves both output FETs off, so the pin floats. In that condition it can be used as a high-impedance input.

Because Ports 1, 2, and 3 have fixed internal pullups they are sometimes called "quasi-bidirectional" ports. When configured as inputs they pull high and will source current (IIL, in the data sheets) when externally pulled low. Port 0, on the other hand, is considered "true" bidirectional, because when configured as an input it floats.

All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

### 7.4.2 Writing to a Port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1). Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pull-up can source about 100 times the current that the normal pull-up can. It should be noted that the internal pull-ups are field-effect transistors, not linear resistors. The pull-up arrangements are shown in Figure 7-6.

In HMOS versions of the 8051, the fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. This transistor will allow the pin to source
A. HMOS Configuration. The enhancement mode transistor is turned on for 2 osc. periods after Q makes a 1-to-0 transition.

B. CHMOS Configuration. pFET 1 is turned on for 2 osc. periods after Q makes a 1-to-0 transition. During this time, pFET 1 also turns on pFET 3 through the inverter to form a latch which holds the 1. pFET 2 is also on.

Figure 7-6. Ports 1 and 3 HMOS and CHMOS Internal Pull-up Configurations. Port 2 is similar except that it holds the strong pullup on while emitting 1s that are address bits. (See text, "Accessing External Memory.")

About 0.25 mA when shorted to ground. In parallel with the fixed pull-up is an enhancement-mode transistor, which is activated during SI whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow the pin to source an additional 30 mA.

In the CHMOS versions, the pull-up consists of three pFETs. It should be noted that an n-channel FET (nFET) is turned on when a logical 1 is applied to its gate, and is turned off when a logical 0 is applied to its gate. A p-channel FET (pFET) is the opposite: it is on when its gate sees a 0, and off when its gate sees a 1.

pFET 1 in Figure 7-6B is the transistor that is turned on for 2 oscillator periods after a 0-to-1 transition in the port latch. While it's on, it turns on pFET 3 (a weak pull-up), through the inverter. This inverter and pFET form a latch which hold the 1.

Note that if the pin is emitting a 1, a negative glitch on the pin from some external source can turn off pFET 3, causing the pin to go into a float state. pFET 2 is a very weak pull-up which is on whenever the pFET is off, in traditional CMOS style. It's only about 1/10 the strength of pFET 3. Its function is to restore a 1 to the pin in the event the pin had a 1 and lost it to a glitch.
7.4.3 Port Loading and Interfacing

The output buffers of Ports 1, 2, and 3 can each drive 4 LS TTL inputs. These ports on HMOS versions can be driven in a normal manner by any TTL or NMOS circuit. Both HMOS and CHMOS pins can be driven by open-collector and open-drain outputs, but note that 0-to-1 transitions will not be fast. In the HMOS device, if the pin is driven by an open collector output, a 0-to-1 transition will have to be driven by the relatively weak depletion mode FET in Figure 7-6(A). In the CHMOS device, an input 0 turns off pull-up pFET3, leaving only the very weak pull-up pFET2 to drive the transition.

Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pull-ups to drive NMOS inputs, except when being used as the ADDRESS/DATA bus.

7.4.4 Read-Modify-Write Feature

Some instructions that read a port read the latch and others read the pin. Which ones do which? The instructions that read the latch rather than the pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called “read-modify-write” instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin:

- **ANL** (logical AND, e.g., ANL P1,A)
- **ORL** (logical OR, e.g., ORL P2,A)
- **XRL** (logical EX-OR, e.g., XRL P3,A)
- **JBC** (Jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL)
- **CPL** (complement bit, e.g., CPL P3.0)
- **INC** (increment, e.g., INC P2)
- **DEC** (decrement, e.g., DEC P2)
- **DJNZ** (decrement and jump if not zero, e.g., DJNZ P3, LABEL)
- **MOV PX,Y,C** (move carry bit to bit Y of Port X)
- **CLR PX,Y** (clear bit Y of Port X)
- **SET PX,Y** (set bit Y of Port X)

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. They read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1.

7.5 ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal PSEN (program store enable) as the read strobe. Accesses to external Data Memory use RD or WR (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetishes from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address (MOVX @DPTR) or an 8-bit address (MOVX @Ri).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. Note that the Port 2 drivers use the strong pullups during the entire time that they are emitting address bits that are 1s. This is during the execution of a MOVX @DPTR instruction. During this time the Port 2 latch (the Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used (MOVX @Ri), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus, in this application the Port 0 pins are not open-drain outputs, and do not require external pull-ups. Signal ALE (address latch enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before WR is activated, and remains there until after WR is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes OFFH to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

External Program Memory is accessed under two conditions:

1) Whenever signal EA is active; or
2) Whenever the program counter (PC) contains a number that is larger than OFFFH (1FFFH for the 8052).

This requires that the ROMless versions have EA wired
low to enable the lower 4K (8K for the 8032) program bytes to be fetched from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function and may not be used for general purpose I/O. During external program fetches they output the high byte of the PC. During this time the Port 2 drivers use the strong pullups to emit PC bits that are 1s.

7.5.1 PSEN

The read strobe for external fetches is PSEN. PSEN is not activated for internal fetches. When the CPU is accessing external Program Memory, PSEN is activated twice every cycle (except during a MOVX instruction) whether or not the byte fetched is actually needed for the current instruction. When PSEN is activated its timing is not the same as RD. A complete RD cycle, including activation and deactivation of ALE and RD, takes 12 oscillator periods. A complete PSEN cycle, including activation and deactivation of ALE and PSEN, takes 6 oscillator periods. The execution sequence for these two types of read cycles are shown in Figure 7-7 for comparison.

7.5.2 ALE

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during fetches from external Program Memory. For that purpose ALE is activated twice every machine cycle. This activation takes place even when the cycle involves no external fetch. The only time an ALE pulse doesn’t come out is during an access to external Data Memory. The first ALE of the second cycle of a MOVX instruction is missing (see Figure 7-7). Consequently, in any system that does not use external Data Memory, ALE is activated at a constant rate of 1/6 the oscillator frequency, and can be used for external clocking or timing purposes.

Figure 7-7. External Program Memory Execution
7.5.3 Overlapping External Program and Data Memory Spaces

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051, the external Program and Data Memory spaces can be combined by ANDing PSEN and RD. A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the PSEN cycle is faster than the RD cycle, the external memory needs to be fast enough to accommodate the PSEN cycle.

7.6 TIMER/COUNTERS

The 8051 has two 16-bit timer/counter registers: Timer 0 and Timer 1. The 8052 has these two plus one more: Timer 2. All three can be configured to operate either as timers or event counters.

In the "timer" function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, TO, T1 or (in the 8052) T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the "timer" or "counter" selection, Timer 0 and Timer 1 have four operating modes from which to select. Timer 2, in the 8052, has three modes of operation: "capture," "auto-reload" and "baud rate generator."

7.6.1 Timer 0 and Timer 1

These timer/counters are present in both the 8051 and the 8052. The "timer" or "counter" function is selected by control bits C/T in the Special Function Register TMOD (Figure 6-8). These two timer/counters have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1, and 2 are the same for both timer/counters. Mode 3 is different. The four operating modes are described below.

**MODE 0**

Putting either Timer into mode 0 makes it look like an 8048 Timer, which is an 8-bit counter with a divide-by-32 prescaler. Figure 7-9 shows the mode 0 operation as it applies to Timer 1.

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON (Figure 7-10). GATE is in TMOD.

<table>
<thead>
<tr>
<th>(MSB)</th>
<th>GATE</th>
<th>C/T</th>
<th>M1</th>
<th>M0:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GATE</td>
<td>Gating control When set, Timer/counter &quot;x&quot; is enabled only while &quot;INTx&quot; pin is high and &quot;TRx&quot; control pin is set. When cleared Timer/&quot;x&quot; is enabled whenever &quot;TRx&quot; control bit is set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C/T</td>
<td>Timer or Counter Selector Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from &quot;Tx&quot; input pin).</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(LSB)</th>
<th>GATE</th>
<th>C/T</th>
<th>M1</th>
<th>M0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **M1** M0: Operating Mode
  - 0 0: MCS-48 Timer "TLx" serves as five-bit prescaler.
  - 0 1: 16-bit Timer/Counter "THx" and "TLx" are cascaded; there is no prescaler.
  - 1 0: 8-bit auto-reload timer-counter "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
  - 1 1: (Timer 0) TLO is an eight-bit timer counter-controlled by the standard Timer 0 control bits.
  - (Timer 1) Timer-counter 1 stopped.

Figure 7-8. TMOD: Timer/Counter Mode Control Register
The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0 and INTO for the corresponding Timer 1 signals in Figure 7-9. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

**MODE 1**

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

**MODE 2**

Mode 2 configures the timer register as an 8-bit counter (TL1) with automatic reload, as shown in Figure 7-11. Overflow from TL1 not only sets TF1, but also reloads TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer/Counter 0.

**MODE 3**

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0.

![Figure 7-9. Timer/Counter 1 Mode 0: 13-bit Counter](image)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF1</td>
<td>TCON.7</td>
<td>Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware</td>
</tr>
<tr>
<td>TR1</td>
<td>TCON.6</td>
<td>Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.</td>
</tr>
<tr>
<td>TF0</td>
<td>TCON.5</td>
<td>Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared by hardware</td>
</tr>
<tr>
<td>TR0</td>
<td>TCON.4</td>
<td>Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.</td>
</tr>
</tbody>
</table>

![Figure 7-10. TCON: Timer/Counter Control Register](image)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF1</td>
<td>TCON.7</td>
<td>Timer 1 overflow Flag. Set by hardware when external interrupt edge detected.</td>
</tr>
<tr>
<td>TR1</td>
<td>TCON.6</td>
<td>Timer 1 Run control bit. Set/cleared by software.</td>
</tr>
<tr>
<td>TF0</td>
<td>TCON.5</td>
<td>Timer 0 overflow Flag. Set by hardware when external interrupt edge detected.</td>
</tr>
<tr>
<td>TR0</td>
<td>TCON.4</td>
<td>Timer 0 Run control bit. Set/cleared by software.</td>
</tr>
<tr>
<td>IE1</td>
<td>TCON.3</td>
<td>Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected.</td>
</tr>
<tr>
<td>IT1</td>
<td>TCON.2</td>
<td>Interrupt 1 Type control bit. Set/cleared by software.</td>
</tr>
<tr>
<td>IE0</td>
<td>TCON.1</td>
<td>Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected.</td>
</tr>
<tr>
<td>IT0</td>
<td>TCON.0</td>
<td>Interrupt 0 Type control bit. Set/cleared by software.</td>
</tr>
</tbody>
</table>
Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 7-12. TL0 uses the Timer 0 control bits: C/T, GATE, TR0, INTO, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer or counter. With Timer 0 in Mode 3, an 8051 can look like it has three timer/counters, and an 8052, like it has four. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

7.6.2 Timer 2

Timer 2 is a 16-bit timer/counter which is present only in the 8052. Like Timers 0 and 1, it can operate either as a timer or as an event counter. This is selected by bit C/T2 in the Special Function Register T2CON (Figure 7-13). It has three operating modes: "capture," "auto-
In the capture mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. (RCAP2L and RCAP2H are new Special Function Registers in the 8052.) In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt.

The capture mode is illustrated in Figure 7-14.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2.

The auto-reload mode is illustrated in Figure 7-15.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It will be described in conjunction with the serial port.

7.7 SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn’t been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Special Function Register SBUF. Writing to SBUF loads the transmit reg-
The serial port can operate in 4 modes:

**Mode 0:** Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

**Mode 1:** 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

**Mode 2:** 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

**Mode 3:** 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

### 7.7.2 Serial Port Control Register

The serial port control and status register is the Special Function Register SCON, shown in Figure 7-16. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupt bits (TI and RI).

---

**7.7.1 Multiprocessor Communications**

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.
### MCS®-51 Architecture

#### 7.7.3 Baud Rates

The baud rate in Mode 0 is fixed:

\[
\text{Mode 0 Baud Rate} = \frac{\text{Oscillator Frequency}}{12}
\]

The baud rate in Mode 2 depends on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is its value on reset), the baud rate is 1/64 the oscillator frequency. If SMOD = 1, the baud rate is 1/32 the oscillator frequency.

\[
\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD}}}{64} \times (\text{Oscillator Frequency})
\]
In the 8051, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate. In the 8052, these baud rates can be determined by Timer 1, or by Timer 2, or by both (one for transmit and the other for receive).

### Using Timer 1 to Generate Baud Rates

When Timer 1 is used as the baud rate generator, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate and the value of SMOD as follows:

**Modes 1, 3 Baud Rate**

\[
\text{Baud Rate} = \frac{2^{\text{SMOD}} x \text{SMOD}}{32} \times \left(\text{Timer 1 Overflow Rate}\right)
\]

The Timer 1 interrupt should be disabled in this application. The Timer itself can be configured for either "timer" or "counter" operation, and in any of its 3 running modes. In the most typical applications, it is configured for "timer" operation, in the auto-reload mode (high nibble of TMOD = 0010B). In that case, the baud rate is given by the formula

**Modes 1, 3 Baud Rate**

\[
\text{Baud Rate} = \frac{2^{\text{SMOD}} \times \text{Oscillator Frequency}}{32} \times \left(12 \times \left(256 - (\text{TH1})\right)\right)
\]

One can achieve very low baud rates with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD = 0001B), and using the Timer 1 interrupt to do a 16-bit software reload.

Figure 7-17 lists various commonly used baud rates and how they can be obtained from Timer 1.

<table>
<thead>
<tr>
<th>BAUD RATE</th>
<th>(f_{osc})</th>
<th>SMOD</th>
<th>C/T</th>
<th>MODE</th>
<th>RELOAD VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODE 0 MAX: 1MHZ</td>
<td>12 MHZ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MODE 1: 375K</td>
<td>12 MHZ</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>MODES 1,3: 62.5K</td>
<td>12 MHZ</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>77H</td>
</tr>
<tr>
<td>19.2K</td>
<td>11.059 MHZ</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>9.6K</td>
<td>11.059 MHZ</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>4.8K</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FAH</td>
</tr>
<tr>
<td>2.4K</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FAH</td>
</tr>
<tr>
<td>1.2K</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FBH</td>
</tr>
<tr>
<td>625.00</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>312.50</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>156.25</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>78.125</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>39.0625</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>19.53125</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>9.765625</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
<tr>
<td>4.8828125</td>
<td>11.059 MHZ</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>FDH</td>
</tr>
</tbody>
</table>

### Using Timer 2 to Generate Baud Rates

In the 8052, Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Figure 7-13). Note then the baud rates for transmit and receive can be simultaneously different. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 7-18.

![Figure 7-18. Timer 2 in Baud Rate Generator Mode](image-url)
The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

Now, the baud rates in Modes 1 and 3 are determined by Timer 2’s overflow rate as follows:

**Modes 1, 3 Baud Rate = \frac{\text{Timer 2 Overflow Rate}}{16}**

The Timer can be configured for either “timer” or “counter” operation. In the most typical applications, it is configured for “timer” operation (C/T2 = 0). “Timer” operation is a little different for Timer 2 when it’s being used as a baud rate generator. Normally as a timer it would increment every machine cycle (thus at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (thus at 1/2 the oscillator frequency). In that case the baud rate is given by the formula

**Modes 1, 3**

**Baud Rate = \frac{\text{Oscillator Frequency}}{32x(65536 \text{-} (RCAP2H, RCAP2L))}**

where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

Timer 2 as a baud rate generator is shown in Figure 7-18. This Figure is valid only if RCLK + TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2, and will not generate an interrupt. Therefore, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt, if desired.

It should be noted that when Timer 2 is running (TR2 = 1) in “timer” function in the baud rate generator mode, one should not try to read or write TH2 or TL2. Under these conditions the Timer is being incremented every state time, and the results of a read or write may not be accurate. The RCAP registers may be read, but shouldn’t be written to, because a write might overlap a reload and cause write and/or reload errors. Turn the Timer off (clear TR2) before accessing the Timer 2 or RCAP registers, in this case.

### 7.7.4 More About Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal at S6P2 also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between “write to SBUF,” and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1 and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeros come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position, is just to the left of the MSB, and all positions to the left of that contain zeros. This condition flags the TX Control block to do one last shift and then deactivate SEND and set T1. Both of these actions occur at S1P1 of the 10th machine cycle after “write to SBUF.”

Reception is initiated by the condition REN = 1 and RI = 0. At S6P2 of the next machine cycle, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

### 7.7.5 More About Mode 1

Ten bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. In the 8051 the baud rate is determined by the Timer 1 overflow rate. In the 8052 it is determined either by the Timer 1 overflow rate, or the Timer 2 overflow rate, or both (one for transmit and the other for receive).

Figure 7-20 shows a simplified functional diagram of the serial port in Mode 1, and associated timings for transmit and receive.
Figure 7-19. Serial Port Mode 0
Figure 7-20. Serial Port Mode 1
TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.
Transmission is initiated by any instruction that uses $\text{SBUF}$ as a destination register. The "write to $\text{SBUF}$" signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to $\text{SBUF}$" signal).

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeros are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate SEND and set TI. This occurs at the 10th divide-by-16 rollover after "write to $\text{SBUF}$."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Reseting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register, (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load $\text{SBUF}$ and RB8, and set RI. The signal to load $\text{SBUF}$ and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

1) $\text{RI} = 0$, and
2) Either $\text{SM2} = 0$, or the received stop bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into $\text{SBUF}$, and RI is activated. At this time, whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

### 7.7.6 More About Modes 2 and 3

Eleven bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency in mode 2. Mode 3 may have a variable baud rate generated from either Timer 1 or 2 depending on the state of TCLK and RCLK.

Figures 7-21 A and B show a functional diagram of the serial port in modes 2 and 3. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses $\text{SBUF}$ as a destination register. The "write to $\text{SBUF}$" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to $\text{SBUF}$" signal.)

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th position of the transmit shift register. The first shift clock an 0 immediately after that. The first shift clock an 0 immediately after that.

The bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register.

At the 7th, 8th and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.
Figure 7-21A. Serial Port Mode 2
Figure 7-21B. Serial Port Mode 3
TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.
As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

1) RI = 0, and
2) Either SM2 = 0 or the received 9th data bit = 1

If either of these conditions is not met, the received frame is irretrievably lost, and RI is not set. If both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

### 7.8 Interrupts

The 8051 provides 5 interrupt sources. The 8052 provides 6. These are shown in Figure 7-22.

The External Interrupts INT0 and INT1 can each be either level-activated or transition-activated, depending on bits IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

The Timer 0 and Timer 1 Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective timer/counter registers (except see Section 7.6.1 for Timer 0 in mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

In the 8052, the Timer 2 Interrupt is generated by the logical OR of TF2 and EXF2. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software.

![Figure 7-22. MCS-51 Interrupt Sources](image)

![Figure 7-23, IE: Interrupt Enable Register](image)
Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Figure 7-23). Note that IE contains also a global disable bit, EA, which disables all interrupts at once.

7.8.1 Priority Level Structure

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP (Figure 7-24). A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP.7</td>
<td>X</td>
<td>reserved</td>
</tr>
<tr>
<td>IP.6</td>
<td>X</td>
<td>reserved</td>
</tr>
<tr>
<td>PT2</td>
<td>IP.5</td>
<td>defines the Timer 2 interrupt priority level, PT2 = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PS</td>
<td>IP.4</td>
<td>defines the Serial Port interrupt priority level, PS = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PT1</td>
<td>IP.3</td>
<td>defines the Timer 1 interrupt priority level, PT1 = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PX1</td>
<td>IP.2</td>
<td>defines the External interrupt 1 priority level, PX1 = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PT0</td>
<td>IP.1</td>
<td>defines the External interrupt 0 priority level, PT0 = 1 programs it to the higher priority level.</td>
</tr>
<tr>
<td>PX0</td>
<td>IP.0</td>
<td>defines the External interrupt 0 priority level, PX0 = 1 programs it to the higher priority level.</td>
</tr>
</tbody>
</table>

Figure 7-24. IP: Interrupt Priority Register

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>PRIORITY WITHIN LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. IEO</td>
<td>(highest)</td>
</tr>
<tr>
<td>2. TF0</td>
<td></td>
</tr>
<tr>
<td>3. IE1</td>
<td></td>
</tr>
<tr>
<td>4. TF1</td>
<td></td>
</tr>
<tr>
<td>5. RI + TI</td>
<td></td>
</tr>
<tr>
<td>6. TF2 + EXF2</td>
<td>(lowest)</td>
</tr>
</tbody>
</table>

Note that the "priority within level" structure is only used to resolve simultaneous requests of the same priority level.

7.8.2 How Interrupts Are Handled

The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.
2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.
3. The instruction in progress is RETI or any access to the IE or IP registers.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed.
before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE or IP, then at least one more instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle, and the values polled are the values that were present at SS2 of the previous machine cycle. Note then that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not still active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new.

The polling cycle/LCALL sequence is illustrated in Figure 7-25.

Note that if an interrupt of higher priority level goes active prior to SS2 of the machine cycle labeled C3 in Figure 7-25, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn’t. It never clears the Serial Port or Timer 2 flags. This has to be done in the user’s software. It clears an external interrupt flag (IEO or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown below.

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEO</td>
<td>0003H</td>
</tr>
<tr>
<td>TF0</td>
<td>000BH</td>
</tr>
<tr>
<td>IE1</td>
<td>0013H</td>
</tr>
<tr>
<td>TF1</td>
<td>001BH</td>
</tr>
<tr>
<td>RI + TI</td>
<td>0023H</td>
</tr>
<tr>
<td>TF2 + EXF2</td>
<td>002BH</td>
</tr>
</tbody>
</table>

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

### 7.8.3 External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit ITI or IT0 in Register TCON. If ITx = 0, external interrupt x is triggered by a detected low at the INTx pin. If ITx = 1, external interrupt x is edge-triggered. In this mode if successive samples of the INTx pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

### 7.8.4 Response Time

The INT0 and INTI levels are inverted and latched into IEO and IE1 at SS2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Figure 7-25 shows interrupt response timings.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt’s service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long, and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 8 cycles.
7.9 SINGLE-STEP OPERATION

The 8051 interrupt structure allows single-step execution with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least one instruction of the interrupted program is executed. One way to use this feature for single-step operation is to program one of the external interrupts (say, INTO) to be level-activated. The service routine for the interrupt will terminate with the following code:

```
JNB P3.2,$ ;WAIT HERE TILL INTO GOES HIGH
JB P3.2,$ ;NOW WAIT HERE TILL IT GOES LOW
RETI ;GO BACK AND EXECUTE ONE INSTRUCTION
```

Now if the INTO pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until INTO is pulsed (from low to high to low). Then it will execute RETI, go back to the task program, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

7.10 RESET

The reset input is the RST pin, which is the input to a Schmitt Trigger.

A reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods), while the oscillator is running. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional). The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

<table>
<thead>
<tr>
<th>REGISTER</th>
<th>CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>0000H</td>
</tr>
<tr>
<td>ACC</td>
<td>00H</td>
</tr>
<tr>
<td>B</td>
<td>00H</td>
</tr>
<tr>
<td>PSW</td>
<td>00H</td>
</tr>
<tr>
<td>SP</td>
<td>07H</td>
</tr>
<tr>
<td>DPTR</td>
<td>0000H</td>
</tr>
<tr>
<td>P0-P3</td>
<td>OFFH</td>
</tr>
<tr>
<td>IP (8051)</td>
<td>XXX00000B</td>
</tr>
<tr>
<td>IP (8052)</td>
<td>XXX00000B</td>
</tr>
<tr>
<td>IE (8051)</td>
<td>0XX00000B</td>
</tr>
<tr>
<td>IE (8052)</td>
<td>0XX00000B</td>
</tr>
<tr>
<td>TMOD</td>
<td>00H</td>
</tr>
<tr>
<td>TCON</td>
<td>00H</td>
</tr>
<tr>
<td>T2CON (8052 only)</td>
<td>00H</td>
</tr>
<tr>
<td>TH0</td>
<td>00H</td>
</tr>
<tr>
<td>TL0</td>
<td>00H</td>
</tr>
<tr>
<td>TH1</td>
<td>00H</td>
</tr>
<tr>
<td>TL1</td>
<td>00H</td>
</tr>
<tr>
<td>TH2</td>
<td>00H</td>
</tr>
<tr>
<td>TL2</td>
<td>00H</td>
</tr>
<tr>
<td>RCAP2H (8052 only)</td>
<td>00H</td>
</tr>
<tr>
<td>RCAP2L (8052 only)</td>
<td>00H</td>
</tr>
<tr>
<td>SCON</td>
<td>00H</td>
</tr>
<tr>
<td>SBUF</td>
<td>Indeterminate</td>
</tr>
<tr>
<td>PCON (HMOS)</td>
<td>0XXXXXXXB</td>
</tr>
<tr>
<td>PCON (CHMOS)</td>
<td>0XXXX000B</td>
</tr>
</tbody>
</table>

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless the part is returning from a reduced power mode of operation.

POWER-ON RESET

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a 10 μF capacitor and to VSS through an 8.2KΩ resistor, providing the VCC risetime does not exceed a millisecond and the oscillator start-up time does not exceed 10 milliseconds. This power-on reset circuit is shown in Figure 7-26. When power comes on, the current drawn by RST commences to charge the capacitor. The voltage at RST is the difference between VCC and the capacitor voltage, and decreases from VCC as the cap charges. The larger the capacitor, the more slowly VRST decreases. VRST must remain above the lower threshold of the Schmitt Trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

7.11 POWER-SAVING MODES OF OPERATION

For applications where power consumption is a critical factor, both the HMOS and CHMOS versions provide reduced power modes of operation. For the CHMOS ver-
sion of the 8051 the reduced power modes, Idle and Power Down, are standard features. In the HMOS versions a reduced power mode is available, but not as a standard feature. The local sales office will provide ordering information for users requiring this feature.

7.11.1 HMOS Power Down Mode

The power down mode in the HMOS devices allows one to reduce VCC to zero while saving the on-chip RAM through a backup supply connected to the RST pin. To use the feature, the user’s system, upon detecting that a power failure is imminent, would interrupt the processor in some manner to transfer relevant data to the on-chip RAM and enable the backup power supply to the RST pin before VCC falls below its operating limit. When power returns, the backup supply needs to stay on long enough to accomplish a reset, and then can be removed so that normal operation can be resumed.

7.11.2 CHMOS Power Reduction Modes

CHMOS versions have two power-reducing modes, Idle and Power Down. The input through which backup power is supplied during these operations is VCC. Figure 7-27 shows the internal circuitry which implements these features. In the Idle mode (IDL = 1), the oscillator continues to run and the Interrupt, Serial Port, and Timer blocks continue to be clocked, but the clock signal is gated off to the CPU. In Power Down (PD = 1), the oscillator is frozen. The Idle and Power Down modes are activated by setting bits in Special Function Register PCON. The address of this register is 87H. Figure 7-28 details its contents.

IDLE MODE

An instruction that sets PCON.0 causes that to be the last instruction executed before going into the Idle mode. In the Idle mode, the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. ALE and PSEN hold at logic high levels. There are two ways to terminate the Idle. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that put the device into Idle.

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMOD</td>
<td>PCON.7</td>
<td>Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.</td>
</tr>
<tr>
<td></td>
<td>PCON.6</td>
<td>(Reserved)</td>
</tr>
<tr>
<td></td>
<td>PCON.5</td>
<td>(Reserved)</td>
</tr>
<tr>
<td></td>
<td>PCON.4</td>
<td>(Reserved)</td>
</tr>
<tr>
<td>GF1</td>
<td>PCON.3</td>
<td>General-purpose flag bit.</td>
</tr>
<tr>
<td>GF0</td>
<td>PCON.2</td>
<td>General-purpose flag bit.</td>
</tr>
<tr>
<td>PD</td>
<td>PCON.1</td>
<td>Power Down bit. Setting this bit activates power down operation.</td>
</tr>
<tr>
<td>IDL</td>
<td>PCON.0</td>
<td>Idle mode bit. Setting this bit activates idle mode operation.</td>
</tr>
</tbody>
</table>

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (00000000).

Figure 7-28. PCON: Power Control Register

Figure 7-27. Idle and Power Down Hardware
during an Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way of terminating the Idle mode is with a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

**POWER DOWN MODE**

An instruction that sets PCON.1 causes that to be the last instruction executed before going into the Power Down mode. In the Power Down mode, the on-chip oscillator is stopped. With the clock frozen, all functions are stopped, but the on-chip RAM and Special Function Registers are held. The port pins output the values held by their respective SFRs. ALE and PSEN output lows.

The only exit from Power Down is a hardware reset. Reset redefines all the SFRs, but does not change the on-chip RAM.

In the Power down mode of operation, VCC can be reduced to minimize power consumption. Care must be taken, however, to ensure that VCC is not reduced before the Power Down mode is invoked, and that VCC is restored to its normal operating level, before the Power Down mode is terminated. The reset that terminates Power Down also frees the oscillator. The reset should not be activated before VCC is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 msec).

**7.12 8751H**

The 8751H is the EPROM member of the MCS-51 family. This means that the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. The 8751H also has a provision for denying external access to the on-chip Program Memory, in order to protect its contents against software piracy.

**7.12.1 Programming the EPROM**

To be programmed, the 8751H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4–P2.6 and PSEN should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for a logic high.) EA/VPP is held normally high, and is pulsed to +21V. While EA/VPP is at 21V, the ALE/PROG pin, which is normally being held high, is pulsed low for 50 msec. Then EA/VPP is returned to high. This setup is shown in Figure 7.29. Detailed timing specifications are provided in the 8751H data sheet.

Note: The EA pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

**7.12.2 Program Verification**

If the program security bit has not been programmed, the on-chip Program Memory can be read out for verification.
purposes, if desired, either during or after the programming operation. The required setup, which is shown in Figure 7.30, is the same as for programming the EPROM except that pin P2.7 is held at TTL low (or used as an active-low read strobe). The address of the Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3. The other Port 2 pins and PSEN are held low. ALE, EA, and RST are held high. The contents of the addressed location will come out on Port 0. External pull-ups are required on Port 0 for this operation.

**7.12.3 Program Memory Security**

The 8751H contains a security bit, which, once programmed, denies electrical access by any external means to the on-chip Program Memory. The setup and procedure for programming the security bit are the same as for normal programming, except that pin P2.6 is held at TTL high. The setup is shown in Figure 7.31. Port 0, Port 1, and pins P2.0-P2.3 of Port 2 may be in any state.

Once the security bit has been programmed, it can be deactivated only by full erasure of the Program Memory. While it is programmed, the internal Program Memory cannot be read out, the device cannot be further programmed, and it cannot execute external program memory. Erasing the EPROM, thus deactivating the security bit, restores the device's full functionality. It can then be re-programmed.

**7.12.4 Erasure Characteristics**

Erasure of the 8751H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter
than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the 8751H to this type of exposure, it is suggested that an opaque label be placed over the window.

Erasure leaves the array in an all 1s state.

7.13 MORE ABOUT THE ON-CHIP OSCILLATOR

7.13.1 HMOS Versions

The on-chip oscillator circuitry for the HMOS (HMOS-I and HMOS-II) members of the MCS-51 family is a single stage linear inverter (Figure 7-32), intended for use as a crystal-controlled, positive reactance oscillator (Figure 7-33). In this application the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

![Figure 7-32. On-Chip Oscillator Circuitry in the HMOS Versions of the MCS-51 Family](image1)

![Figure 7-33. Using the HMOS On-Chip Oscillator](image2)
The crystal specifications and capacitance values (C1 and C2 in Figure 7-33) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. A ceramic resonator can be used in place of the crystal in cost-sensitive applications. When a ceramic resonator is used, C1 and C2 are normally selected to be of somewhat higher values, typically, 47 pF. The manufacturer of the ceramic resonator should be consulted for recommendations on the values of these capacitors.

A more in-depth discussion of crystal specifications, ceramic resonators, and the selection of values for C1 and C2 can be found in Application Note AP-155, “Oscillators for Microcontrollers,” which is included in this manual.

To drive the HMOS parts with an external clock source, apply the external clock signal to XTAL2, and ground XTAL1, as shown in Figure 7-34. A pull-up resistor may be used (to increase noise margin), but is optional if VOH of the driving gate exceeds the VIHMIN specification of XTAL2.

7.13.2 CHMOS

The on-chip oscillator circuitry for the 80C51, shown in Figure 7-35, consists of a single stage linear inverter intended for use as a crystal-controlled, positive reactance oscillator in the same manner as the HMOS parts. However, there are some important differences.

One difference is that the 80C51 is able to turn off its oscillator under software control (by writing a 1 to the PD bit in PCON). Another difference is that in the 80C51 the internal clocking circuitry is driven by the signal at XTAL1, whereas in the HMOS versions it is by the signal at XTAL2.

The feedback resistor Rf in Figure 7-35 consists of paralleled n- and p-channel FETs controlled by the PD bit, such that Rf is opened when PD = 1. The diodes D1 and D2, which act as clamps to VCC and VSS, are parasitic to the Rf FETs.
The oscillator can be used with the same external components as the HMOS versions, as shown in Figure 7-36. Typically, $C_1 = C_2 = 30 \text{ pF}$ when the feedback element is a quartz crystal, and $C_1 = C_2 = 47 \text{ pF}$ when a ceramic resonator is used.

To drive the CHMOS parts with an external clock source, apply the external clock signal to XTAL1, and leave XTAL2 float, as shown in Figure 7-37.

The reason for this change from the way the HMOS part is driven can be seen by comparing Figures 7-32 and 7-35. In the HMOS devices the internal timing circuits are driven by the signal at XTAL2. In the CHMOS devices the internal timing circuits are driven by the signal at XTAL1.

**Figure 7-36. Using the CHMOS On-Chip Oscillator**

**7.14 INTERNAL TIMING**

Figures 7-38 through 7-41 show when the various strobe and port signals are clocked internally. The figures do not show rise and fall times of the signals, nor do they show propagation delays between the XTAL2 signal and events at other pins.

Rise and fall times are dependent on the external loading that each pin must drive. They are often taken to be something in the neighborhood of 10 nsec, measured between 0.8V and 2.0V.

Propagation delays are different for different pins. For a given pin they vary with pin loading, temperature, VCC, and manufacturing lot. If the XTAL2 waveform is taken as the timing reference, prop delays may vary from 25 to 125 nsec.

The AC Timings section of the data sheets do not reference any timing to the XTAL2 waveform. Rather, they relate the critical edges of control and input signals to each other. The timings published in the data sheets include the effects of propagation delays under the specified test conditions.

**7.15 MCS-51 PIN DESCRIPTIONS**

**VCC:** Supply voltage.

**VSS:** Circuit ground potential.

**Port 0:** Port 0 is an 8-bit open drain bidirectional I/O port. As an open drain output port it can sink 8 LS TTL loads. Port 0 pins that have 1s written to them float, and in that state will function as high-impedance inputs. Port 0 is also
the multiplexed low-order address and data bus during accesses to external memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also emits code bytes during program verification. In that application, external pullups are required.

**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pullups. The port 1 output buffers can sink/source 4 LS TTL loads. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

In the 8052, pins P1.0 and P1.1 also serve the alternate functions of T2 and T2EX. T2 is the Timer 2 external input. T2EX is the input through which a Timer 2 "capture" is triggered.

**Port 2:** Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL loads. Port 2 emits the high-order address byte during accesses to external memory that use 16-bit addresses. In this application it uses the strong internal pullups when emitting 1s. Port 2 also receives the high-order address and control bits during 8751H programming and verification, and during program verification in the 8051AH.

**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

<table>
<thead>
<tr>
<th>PORT PIN</th>
<th>ALTERNATE FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RXD (serial input port)</td>
</tr>
<tr>
<td>P3.1</td>
<td>TXD (serial output port)</td>
</tr>
<tr>
<td>P3.2</td>
<td>INT0 (external interrupt 0)</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 (external interrupt 1)</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 (Timer 0 external input)</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 (Timer 1 external input)</td>
</tr>
<tr>
<td>P3.6</td>
<td>WR (external data memory write strobe)</td>
</tr>
<tr>
<td>P3.7</td>
<td>RD (external data memory read strobe)</td>
</tr>
</tbody>
</table>

The Port 3 output buffers can source/sink 4 LS TTL loads.

**RST:** Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

**ALE/PROG:** Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE is emitted at a constant rate of 1/6 of the oscillator frequency, for external timing or clocking purposes, even when there are no accesses to external memory. (However, one ALE pulse is skipped during each access to external **Data Memory**.) This pin is also the program pulse input (PROG) during EPROM programming.

**PSEN:** Program Store Enable is the read strobe to external Program Memory. When the device is executing out of external Program Memory, PSEN is activated twice each machine cycle (except that two PSEN activations are skipped during accesses to external **Data Memory**) PSEN is not activated when the device is executing out of internal Program Memory.

**EA/VPP:** When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH in the 8051AH, or 1FFFH in the 8052). Holding EA low forces the CPU to execute out of external memory regardless of the Program Counter value. In the 8031AH and 8032, EA must be externally wired low. In the 8751H, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

**XTAL1:** Input to the inverting oscillator amplifier.

**XTAL2:** Output from the inverting oscillator amplifier.
Figure 7-38. External Program Memory Fetches

Figure 7-39. External Data Memory Read Cycle
Figure 7-40. External Data Memory Write Cycle

Figure 7-41. Port Operation
CHAPTER 8
MCS®-51 INSTRUCTION SET

8.0 INTRODUCTION
The MCS®-51 instruction set includes 111 instructions, 49 of which are single-byte, 45 two-byte and 17 three byte. The instruction op code format consists of a function mnemonic followed by a "destination, source" operand field. This field specifies the data type and addressing method(s) to be used.

8.1 FUNCTIONAL OVERVIEW
The MCS-51 instruction set is divided into four functional groups:
- Data Transfer
- Arithmetic
- Logic
- Control Transfer

8.1.1 Data Transfer
Data transfer operations are divided into three classes:
- General Purpose
- Accumulator-Specific
- Address-Object

None of these operations affect the PSW flag settings except a POP or MOV directly to the PSW.

GENERAL-PURPOSE TRANSFERS
- MOV performs a bit or a byte transfer from the source operand to the destination operand.
- PUSH increments the SP register and then transfers a byte from the source operand to the stack location currently addressed by SP.
- POP transfers a byte operand from the stack location addressed by SP to the destination operand and then decrements SP.

ACCUMULATOR SPECIFIC TRANSFERS
- XCH exchanges the byte source operand with register A (accumulator).
- XCHD exchanges the low-order nibble of the byte source operand with the low-order nibble of A.
- MOVX performs a byte move between the External Data Memory and the accumulator. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).
- MOV C moves a byte from Program memory to the accumulator. The operand in A is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to the accumulator.

ADDRESS-OBJECT TRANSFER
- MOV DPTR, #data loads 16-bits of immediate data into a pair of destination registers, DPH and DPL.

8.1.2 Arithmetic
The 8051 has four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag, however, permits the addition and subtraction operation to serve for both unsigned and signed binary integers. Arithmetic can also be performed directly on packed decimal (BCD) representations.

ADDITION
- INC (increment) adds one to the source operand and puts the result in the operand.
- ADD adds A to the source operand and returns the result to A.
- ADDC (add with Carry) adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.
- DA (decimal-add-adjust for BCD addition) corrects the sum which results from the binary addition of two two-digit decimal operands. The packed decimal sum formed by DA is returned to A. CY is set if the BCD result is greater than 99; otherwise, it is cleared.

SUBTRACTION
- SUBB (subtract with borrow) subtracts the second source operand from the first operand (the accumulator), subtracts one (1) if CY is set and returns the result to A.
- DEC (decrement) subtracts one (1) from the source operand and returns the result to the operand.
MULTIPLICATION

- MUL performs an unsigned multiplication of the A register by the B register, returning a double-byte result. A receives the low-order byte, B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is non-zero. CY is cleared. AC is unaffected.

DIVISION

- DIV performs an unsigned division of the A register by the B register and returns the integer quotient to A and returns the fractional remainder to the B register. Division by zero leaves indeterminate data in registers A and B and sets OV; otherwise OV is cleared. CY is cleared. AC is unaffected.

Unless otherwise stated in the above descriptions, the flags of PSW are affected as follows:

- CY is set if the operation causes a carry to or from the resulting high-order bit. Otherwise CY is cleared.

- AC is set if the operation results in a carry from the low-order four bits of the result (during addition), or a borrow from the high-order bits to the low-order bits (during subtraction); otherwise AC is cleared.

- OV is set if the operation results in a carry to the high-order bit of the result but not a carry from the high-order bit, or vice versa; otherwise OV is cleared. OV is used in two's-complement arithmetic, because it is set when the signed result cannot be represented in 8 bits.

- P is set if the modulo 2 sum of the eight bits in the accumulator is 1 (odd parity); otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.

8.1.3 Logic

The 8051 performs basic logic operations on both bit and byte operands.

SINGLE-OPERAND OPERATIONS

- CLR sets A or any directly addressable bit to zero (0).
- SETB sets any directly addressable bit to one (1).
- CPL is used to complement the contents of the A register without affecting any flags, or any directly addressable bit location.

- RL, RLC, RR, RRC, SWAP are the five rotate operations that can be performed on A. RL, rotate left, RR, rotate right, RLC, rotate left through C, RRC, rotate right through C, and SWAP, rotate left four. For RLC and RRC the CY flag becomes equal to the last bit rotated out. SWAP rotates A left four places to exchange bits 3 through 0 with bits 7 through 4.

TWO-OPERAND OPERATIONS

- ANL performs bitwise logical and of with two source operands (for both bit and byte operands) and returns the result to the location of the first operand.

- ORL performs bitwise logical or of two source operands (for both bit and byte operands) and returns the result of the location of the first operand.

- XRL performs bitwise logical or of two source operands (byte operands) and returns the result to the location of the first operand.

8.1.4 Control Transfer

There are three classes of control transfer operations: unconditional calls, returns and jumps; conditional jumps; and interrupts. All control transfer operations cause, some upon a specific condition, the program execution to continue at a non-sequential location in program memory.

UNCONDITIONAL CALLS, RETURNS AND JUMPS

Unconditional calls, returns and jumps transfer control from the current value of the Program Counter to the target address. Both direct and indirect transfers are supported.

- ACALL and LCALL push the address of the next instruction onto the stack and then transfer control to the target address. ACALL is a 2-byte instruction used when the target address is in the current 2K page. LCALL is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. an 11 bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K page then the call will be made to the next page since the PC will have been incremented to the next instruction prior to execution.
MCS®-51 INSTRUCTION SET

• RET transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.

• AJMP, LJMP and SJMP transfer control to the target operand. The operation of AJMP and LJMP are analogous to ACALL and LCALL. The SJMP (short jump) instruction provides for transfers within a 256-byte range centered about the starting address of the next instruction (-128 to +127).

• JMP @A+DPTR performs a jump relative to the DPTR register. The operand in A is used as the offset (0-255) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the Program Memory space.

CONDITIONAL JUMPS

Conditional jumps perform a jump contingent upon a specific condition. The destination will be within a 256-byte range centered about the starting address of the next instruction (-128 to +127).

• JZ performs a jump if the accumulator is zero.

• JNZ performs a jump if the accumulator is not zero.

• JC performs a jump if the carry flag is set.

• JNC performs a jump if the carry flag is not set.

• JB performs a jump if the Direct Addressed bit is set.

• JNB performs a jump if the Direct Addressed bit is not set.

• JBC performs a jump if the Direct Addressed bit is set and then clears the Direct Addressed bit.

• CJNE compares the first operand to the second operand and performs a jump if they are not equal. CY is set if the first operand is less than the second operand; otherwise it is cleared. Comparisons can be made between A directly addressable bytes in Internal Data Memory or between an immediate value and either A, a register in the selected Register Bank, or a Register-Indirect addressed byte of Internal RAM.

• DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The source operand of the DJNZ instruction may be any byte in the Internal Data Memory. Either Direct or Register Addressing may be used to address the source operand.

INTERRUPT RETURNS

• RETI transfers control as does RET, but additionally enables interrupts of the current priority level.

8.2 INSTRUCTION DEFINITIONS

Each of the 51 basic MCS-51 operations, ordered alphabetically according to the operation mnemonic are described beginning page 8-8.

A brief example of how the instruction might be used is given as well as its effect on the PSW flags. The number of bytes and machine cycles required, the binary machine-language encoding, and a symbolic description or restatement of the function is also provided.

Note: Only the carry, auxiliary-carry, and overflow flags are discussed. The parity bit is computed after every instruction cycle that alters the accumulator. Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by bit-manipulation.

For details on the MCS-51 assembler, ASM51, refer to the MCS-51 Macro Assembler User's Guide, publication number 9800937.

Table 8-1 summarized the MCS®-51 instruction set.
## Interrupt Response Time:

To finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (3 to 7 μs @ 12 MHz).

### INSTRUCTIONS THAT AFFECT FLAG SETTINGS

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>FLAG</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>C</td>
<td>Add register to Accumulator</td>
</tr>
<tr>
<td>ADDC</td>
<td>O</td>
<td>Add direct byte to Accumulator</td>
</tr>
<tr>
<td>SUBB</td>
<td>X</td>
<td>Subtract register from Acc with borrow</td>
</tr>
<tr>
<td>MUL</td>
<td>X</td>
<td>Multiply A &amp; B</td>
</tr>
<tr>
<td>DIV</td>
<td>X</td>
<td>Divide A by B</td>
</tr>
<tr>
<td>DA</td>
<td>X</td>
<td>Decimal Adjust</td>
</tr>
<tr>
<td>RRC</td>
<td>X</td>
<td>Rotate Right Carry</td>
</tr>
<tr>
<td>RLC</td>
<td>X</td>
<td>Rotate Left Carry</td>
</tr>
<tr>
<td>SETBC</td>
<td>X</td>
<td>Set Carry Bit</td>
</tr>
</tbody>
</table>

Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

### ARITHMETIC OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A,Rn</td>
<td>Add register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ADD A,direct</td>
<td>Add direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADD A,@Ri</td>
<td>Add indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ADD A,#data</td>
<td>Add immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A,Rn</td>
<td>Add register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A,direct</td>
<td>Add direct byte to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A,@Ri</td>
<td>Add indirect RAM to Accumulator with Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A,#data</td>
<td>Add immediate data to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>SUBB A,Rn</td>
<td>Subtract register from Acc with borrow</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SUBB A,direct</td>
<td>Subtract direct byte from Acc with borrow</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

Note: All mnemonics copyrighted ©Intel Corporation 1980.
**Table 8-1. 8051 Instruction Set Summary (Continued)**

<table>
<thead>
<tr>
<th>LOGICAL OPERATIONS</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL A,Rn</td>
<td>AND register to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ANL A,direct</td>
<td>AND direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ANL A,@Ri</td>
<td>AND indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ANL A,#data</td>
<td>AND immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ANL direct,A</td>
<td>AND Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ANL direct,#data</td>
<td>AND immediate data to direct byte</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>ORL A,Rn</td>
<td>OR register to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ORL A,direct</td>
<td>OR direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ORL A,@Ri</td>
<td>OR indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ORL A,#data</td>
<td>OR immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ORL direct,A</td>
<td>OR Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ORL direct,#data</td>
<td>OR immediate data to direct byte</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>XRL A,Rn</td>
<td>Exclusive-OR register to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>XRL A,direct</td>
<td>Exclusive-OR direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOGICAL OPERATIONS Cont.</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRL A,@Ri</td>
<td>Exclusive-OR indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>XRL A,#data</td>
<td>Exclusive-OR immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>XRL direct,A</td>
<td>Exclusive-OR Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>XRL direct,#data</td>
<td>Exclusive-OR immediate data to direct byte</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate Accumulator Left</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate Accumulator Left through the Carry</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate Accumulator Right</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate Accumulator Right through the Carry</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles within the Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

All mnemonics copyrighted ©Intel Corporation 1980
### Table 8-1. 8051 Instruction Set Summary (Continued)

<table>
<thead>
<tr>
<th>DATA TRANSFER</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Rn</td>
<td>Move register to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV A,direct</td>
<td>Move direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV A,@Ri</td>
<td>Move indirect RAM to Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV A,#data</td>
<td>Move immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV Rn,A</td>
<td>Move Accumulator to register</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV Rn,direct</td>
<td>Move direct byte to register</td>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOV Rn,#data</td>
<td>Move immediate data to register</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV direct,A</td>
<td>Move Accumulator to direct byte</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV direct,Rn</td>
<td>Move register to direct byte</td>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOV direct,direct</td>
<td>Move direct byte to direct byte</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOV direct,@Ri</td>
<td>Move indirect RAM to direct byte</td>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOV direct,#data</td>
<td>Move immediate data to direct byte</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOV @Ri,A</td>
<td>Move Accumulator to indirect RAM</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MOV @Ri,direct</td>
<td>Move direct byte to indirect RAM</td>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOV @Ri,#data</td>
<td>Move immediate data to indirect RAM</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA TRANSFER Cont.</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV DPTR,#data</td>
<td>Load Data Pointer with a 16-bit constant</td>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOVC A,@A+DPTR</td>
<td>Move Code byte relative to DPTR to Acc</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOVC A,@A+PC</td>
<td>Move Code byte relative to PC to Acc</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOVX A,@Ri</td>
<td>Move External RAM (8-bit addr) to Acc</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOVX A,@DPTR</td>
<td>Move External RAM (16-bit addr) to Acc</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOVX @Ri,A</td>
<td>Move Acc to External RAM (8-bit addr)</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>MOVX @DPTR,A</td>
<td>Move Acc to External RAM (16-bit addr)</td>
<td>1</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>PUSH direct</td>
<td>Push direct byte onto stack</td>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>POP direct</td>
<td>Pop direct byte from stack</td>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>XCH A,Rn</td>
<td>Exchange register with Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>XCH A,direct</td>
<td>Exchange direct byte with Accumulator</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>XCH A,@Ri</td>
<td>Exchange indirect RAM with Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>XCHD A,@Ri</td>
<td>Exchange low-order Digit indirect RAM with Accumulator</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

All mnemonics copyrighted ©Intel Corporation 1980
## 8051 Instruction Set Summary (Continued)

### Table 8-1. 8051 Instruction Set Summary (Continued)

#### BOOLEAN VARIABLE MANIPULATION

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Oscillator Period</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>Clear Carry</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>CLR bit</td>
<td>Clear direct bit</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SETB C</td>
<td>Set Carry</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SETB bit</td>
<td>Set direct bit</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement Carry</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>CPL bit</td>
<td>Complement direct bit</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>ANL C,bit</td>
<td>AND direct bit</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>ANL C,/bit</td>
<td>AND complement of direct bit to Carry</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>ORL C,bit</td>
<td>OR direct bit</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>ORL C,/bit</td>
<td>OR complement of direct bit to Carry</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>MOV C,bit</td>
<td>Move direct bit</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>MOV bit,C</td>
<td>Move Carry to direct bit</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>JC rel</td>
<td>Jump if Carry is set</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>JNC rel</td>
<td>Jump if Carry not set</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>JB bit,rel</td>
<td>Jump if direct Bit is set</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>JNB bit,rel</td>
<td>Jump if direct Bit is Not set</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>JBC bit,rel</td>
<td>Jump if direct Bit is set &amp; clear bit</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

#### PROGRAM BRANCHING Cont.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Oscillator Period</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETI</td>
<td>Return from interrupt</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>AJMP addr11</td>
<td>Absolute Jump</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>LJMP addr16</td>
<td>Long Jump</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>SJMP rel</td>
<td>Short Jump (relative addr)</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>JMP @A+DPTR</td>
<td>Jump indirect relative to the DPTR</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>JZ rel</td>
<td>Jump if Accumulator is Zero</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>JNZ rel</td>
<td>Jump if Accumulator is Not Zero</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>CJNE A, direct, rel</td>
<td>Compare direct byte to Acc and Jump If Not Equal</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>CJNE A, #data, rel</td>
<td>Compare immediate to Acc and Jump If Not Equal</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>CJNE Rn, #data, rel</td>
<td>Compare immediate to register and Jump If Not Equal</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>CJNE @Ri, #data, rel</td>
<td>Compare immediate to indirect and Jump If Not Equal</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>DJNZ Rn, rel</td>
<td>Decrement register and Jump if Not Zero</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>DJNZ direct, rel</td>
<td>Decrement direct byte and Jump if Not Zero</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

All mnemonics copyrighted ©Intel Corporation 1980
ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2
Cycles: 2

Encoding: 

<table>
<thead>
<tr>
<th>a10</th>
<th>a9</th>
<th>a8</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

Operation: ACALL

(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(PC10-0) ← page address
ADD A,<src-byte>

**Function:** Add

**Description:** ADD adds the byte variable indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

```
ADD A,R0
```

will leave 6DH (01101101B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

**ADD A,Rn**

| Bytes: | 1 |
| Cycles: | 1 |

**Encoding:**

```
0 0 1 0 1 r r r
```

**Operation:**

```
ADD (A) ← (A) + (Rn)
```

**ADD A,direct**

| Bytes: | 2 |
| Cycles: | 1 |

**Encoding:**

```
0 0 1 0 0 1 0 1  | direct address
```

**Operation:**

```
ADD (A) ← (A) + (direct)
```
ADD A, @Ri
Bytes: 1
Cycles: 1
Encoding: 0 0 1 0 0 1 1 i
Operation: ADD
(A) ← (A) + ((Ri))

ADD A, #data
Bytes: 2
Cycles: 1
Encoding: 0 0 1 0 0 1 0 0 immediate data
Operation: ADD
(A) ← (A) + #data
ADDCA, <src-byte>

**Function:** Add with Carry

**Description:** ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carryout of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction, ADDC A,R0

will leave 6EH (01101110B) in the accumulator with AC cleared and both the carry flag and OV set to 1.

**ADDCA,Rn**

| Bytes: | 1 |
| Cycles: | 1 |

| Encoding: | 001111rrr |

| Operation: | ADDC (A)← (A) + (C) + (Rn) |

**ADDCA,direct**

| Bytes: | 2 |
| Cycles: | 1 |

| Encoding: | 001110101 | direct address |

| Operation: | ADDC (A)← (A) + (C) + (direct) |
ADDC A,@Ri

Bytes: 1
Cycles: 1

Encoding: 0 0 1 1 0 1 1 i

Operation: ADDC  
(A)←(A) + (C) + ((Ri))

ADDC A,#data

Bytes: 2
Cycles: 1

Encoding: 0 0 1 1 0 1 0 0 immediate data

Operation: ADDC  
(A)←(A) + (C) + #data

AJMP addr11

Function: Absolute Jump
Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label “JMPADR” is at program memory location 0123H. The instruction, AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2
Cycles: 2

Encoding: a10 a9 a8 0 0 0 0 1 a7 a6 a5 a4 a3 a2 a1 a0

Operation: AJMP  
(PC)←(PC) + 2  
(PC10-0)← page address
ANL <dest-byte>, <src-byte>

**Function:** Logical-AND for byte variables

**Description:** ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.*

**Example:** If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

ANL A, R0

will leave 41H (01000001B) in the accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time. The instruction,

ANL P1, #01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A, Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

```
0 1 0 1 1 r r r
```

**Operation:**

ANL

(A) ← (A) ^ (Rn)
**MCS®-51 INSTRUCTION SET**

ANL A, direct

- **Bytes:** 2
- **Cycles:** 1

**Encoding:**

\[
\begin{array}{c}
0 \\
1 \\
0 \\
1 \\
0 \\
1 \\
1
\end{array}
\]

**Operation:**

\[
\text{ANL} \quad (A) \leftarrow (A) \land (\text{direct})
\]

ANL A, @Ri

- **Bytes:** 1
- **Cycles:** 1

**Encoding:**

\[
\begin{array}{c}
0 \\
1 \\
0 \\
1 \\
0 \\
1 \\
1
\end{array}
\]

**Operation:**

\[
\text{ANL} \quad (A) \leftarrow (A) \land ((\text{Ri}))
\]

ANL A, #data

- **Bytes:** 2
- **Cycles:** 1

**Encoding:**

\[
\begin{array}{c}
0 \\
1 \\
0 \\
1 \\
0 \\
1 \\
0
\end{array}
\]

**Operation:**

\[
\text{ANL} \quad (A) \leftarrow (A) \land \#\text{data}
\]

ANL direct, A

- **Bytes:** 2
- **Cycles:** 1

**Encoding:**

\[
\begin{array}{c}
0 \\
1 \\
0 \\
1 \\
0 \\
0 \\
1
\end{array}
\]

**Operation:**

\[
\text{ANL} \quad (\text{direct}) \leftarrow (\text{direct}) \land (A)
\]

ANL direct, #data

- **Bytes:** 3
- **Cycles:** 2

**Encoding:**

\[
\begin{array}{c}
0 \\
1 \\
0 \\
1 \\
0 \\
1 \\
1
\end{array}
\]

**Operation:**

\[
\text{ANL} \quad (\text{direct}) \leftarrow (\text{direct}) \land \#\text{data}
\]

8-14
MCS™-51 INSTRUCTION SET

ANL  C, <src-bit>

**Function:** Logical-AND for bit variables

**Description:** If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

**Example:**

Only direct bit addressing is allowed for the source operand.

Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

MOV C,P1.0 ;LOAD CARRY WITH INPUT PIN STATE
ANL C,ACC.7 ;AND CARRY WITH ACCUM. BIT 7
ANL C,/OV ;AND WITH INVERSE OF OVERFLOW FLAG

**ANL C,bit**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

```
1 0 0 0 | 0 0 1 0
```

**Operation:**

```
ANL (C) ← (C) ∧ (bit)
```

**ANL C,/bit**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

```
1 0 1 1 | 0 0 0 0
```

**Operation:**

```
ANL (C) ← (C) ∧ ¬ (bit)
```

CJNE  <dest-byte>,<src-byte>, rel

**Function:** Compare and Jump if Not Equal.

**Description:** CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.
The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
CJNE R7,#60H, NOT_EQ
; NOT_EQ:       JC REQ_LOW
;            ; R7 = 60H.
;            ; IF R7 < 60H.
;            ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to port 1 is also 34H, then the instruction,

```
WAIT: CJNE A,P1,WAIT
```
clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

**CJNE A,direct,rel**

| Bytes: | 3 |
| Cycles: | 2 |

```
1011 0101
```

**Encoding:**

<table>
<thead>
<tr>
<th>(PC)</th>
<th>(PC) + 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PC)</td>
<td>(PC) + relative offset</td>
</tr>
</tbody>
</table>

**Operation:**

IF (A) <> (direct) THEN

IF (A) < (direct) THEN

(C) ← 1

ELSE

(C) ← 0

8-16
MCS®-51 INSTRUCTION SET

CJNE A,#data,rel
Bytes: 3
Cycles: 2
Encoding: 1011 0100 immediate data rel. address
Operation:
(PC) ← (PC) + 3
IF (A) <> data
THEN
(PC) ← (PC) + relative offset
IF (A) < data
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE Rn,#data,rel
Bytes: 3
Cycles: 2
Encoding: 1011 1rrr immediate data rel. address
Operation:
(PC) ← (PC) + 3
IF (Rn) <> data
THEN
(PC) ← (PC) + relative offset
IF (Rn) < data
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE @Ri,#data,rel
Bytes: 3
Cycles: 2
Encoding: 1011 011i immediate data rel. address
Operation:
(PC) ← (PC) + 3
IF ((Ri)) <> data
THEN
(PC) ← (PC) + relative offset
IF ((Ri)) < data
THEN
(C) ← 1
ELSE
(C) ← 0

8-17
### CLR A

<table>
<thead>
<tr>
<th>Function:</th>
<th>Clear Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The accumulator is cleared (all bits set to zero). No flags are affected.</td>
</tr>
<tr>
<td>Example:</td>
<td>The accumulator contains 5CH (01011100B). The instruction,</td>
</tr>
<tr>
<td></td>
<td>CLR A</td>
</tr>
<tr>
<td>Bytes:</td>
<td>1</td>
</tr>
<tr>
<td>Cycles:</td>
<td>1</td>
</tr>
<tr>
<td>Encoding:</td>
<td>11100100</td>
</tr>
<tr>
<td>Operation:</td>
<td>CLR (A) ← 0</td>
</tr>
</tbody>
</table>

### CLR bit

<table>
<thead>
<tr>
<th>Function:</th>
<th>Clear bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.</td>
</tr>
<tr>
<td>Example:</td>
<td>Port 1 has previously been written with 5DH (01011101B). The instruction,</td>
</tr>
<tr>
<td></td>
<td>CLR P1.2</td>
</tr>
<tr>
<td>Bytes:</td>
<td>1</td>
</tr>
<tr>
<td>Cycles:</td>
<td>1</td>
</tr>
<tr>
<td>Encoding:</td>
<td>1100011</td>
</tr>
<tr>
<td>Operation:</td>
<td>CLR (C) ← 0</td>
</tr>
</tbody>
</table>

### CLR C

| Bytes: | 1 |
| Cycles: | 1 |
| Encoding: | 1100011 |
| Operation: | CLR (C) ← 0 |

### CLR bit

| Bytes: | 2 |
| Cycles: | 1 |
| Encoding: | 1100010 | bit address |
| Operation: | CLR (bit) ← 0 |
CPL A

Function: Complement Accumulator
Description: Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice-versa. No flags are affected.
Example: The accumulator contains 5CH (01011100B). The instruction,

```
CPL A
```
will leave the accumulator set to 0A3H (10100011B).

Bytes: 1
Cycles: 1

Encoding: 11110100

Operation: \( \text{CPL} \quad (A) \rightarrow \neg (A) \)

CPL bit

Function: Complement bit
Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

```
CPL P1.1
CPL P1.2
```
will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1
Cycles: 1

Encoding: 10110011

Operation: \( \text{CPL} \quad (C) \rightarrow \neg (C) \)
CPL bit
Bytes: 2
Cycles: 1
Encoding: 1 0 1 1 0 0 1 0  bit address
Operation: CPL
(bit)←¬(bit)

DA A

Function: Decimal-adjust Accumulator for Addition
Description: DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.
Example: The accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence,

```
ADDC A,R3
DA A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110B) in the accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the accumulator to the value 24H (00101000B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD A,#99H
DA A
```

will leave the carry set and 29H in the accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 - 1 = 29.

Bytes: 1
Cycles: 1

Encoding: 1 1 0 1 0 1 0 0

Operation: DA
-contents of Accumulator are BCD
IF \[[(A_{3-0}) > 9] \lor [(AC) = 1]\]
THEN \((A_{3-0}) \leftarrow (A_{3-0}) + 6\)
AND
IF \[[(A_{7-4}) > 9] \lor [(C) = 1]\]
THEN \((A_{7-4}) \leftarrow (A_{7-4}) + 6\)
DEC byte

**Function:** Decrement

**Description:** The variable indicated is decremented by 1. An original value of 00H will underflow to OFFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

**Example:** Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC @R0
DEC R0
DEC @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to OFFH and 3FH.

**DEC A**

- **Bytes:** 1
- **Cycles:** 1

**Encoding:**

```
0 0 0 1 0 1 0 0
```

**Operation:**

```
DEC (A) ← (A) - 1
```

**DEC Rn**

- **Bytes:** 1
- **Cycles:** 1

**Encoding:**

```
0 0 0 1 1 r r r r
```

**Operation:**

```
DEC (Rn) ← (Rn) - 1
```
MCS®-51 INSTRUCTION SET

DEC  direct
Bytes: 2
Cycles: 1
Encoding: 0 0 0 1 0 1 0 1 ☐
Operation: DEC
(direct) ← (direct) − 1

DEC @Ri
Bytes: 1
Cycles: 1
Encoding: 0 0 0 1 0 1 1 i
Operation: DEC
((Ri)) ← ((Ri)) − 1

DIV  AB

Function: Divide
Description: DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,
DIV AB
will leave 13 in the accumulator (0DH or 0001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 × 18) + 17. Carry and OV will both be cleared.

Bytes: 1
Cycles: 4
Encoding: 1 0 0 0 0 1 0 0
Operation: DIV
(A)15-8 ← (A) / (B)
(B)7-0 ← (A) / (B)
**DJNZ** \(<\text{byte}>,<\text{rel-addr}>\)

**Function:** Decrement and Jump if Not Zero

**Description:** DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to OFFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

\[
\begin{align*}
\text{DJNZ} & \quad 40H,\text{LABEL}_1 \\
\text{DJNZ} & \quad 50H,\text{LABEL}_2 \\
\text{DJNZ} & \quad 60H,\text{LABEL}_3
\end{align*}
\]

will cause a jump to the instruction at label \(\text{LABEL}_2\) with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

\[
\begin{align*}
\text{MOV} & \quad \text{R2,}#8 \\
\text{TOGGLE:} & \quad \text{CPL P1.7} \\
\text{DJNZ} & \quad \text{R2,TOGGLE}
\end{align*}
\]

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.
DJNZ  Rn,rel

Bytes: 2
Cycles: 2

Encoding: 1101 1rrr  direct address

Operation: DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) - 1
IF (Rn) > 0 or (Rn) < 0
THEN
(PC) ← (PC) + rel

DJNZ  direct,rel

Bytes: 3
Cycles: 2

Encoding: 1101 0101  direct address  rel. address

Operation: DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) - 1
IF (direct) > 0 or (direct) < 0
THEN
(PC) ← (PC) + rel

INC  <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will
overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as
the original port data will be read from the output data latch, not the input
pins.
Example: Register 0 contains 7EH (0111111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

INC @R0
INC R0
INC @R0

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A
Bytes: 1
Cycles: 1

Encoding: 0 0 0 0 0 1 0 0

Operation: INC (A) ← (A) + 1

INC Rn
Bytes: 1
Cycles: 1

Encoding: 0 0 0 0 1 r r r

Operation: INC (Rn) ← (Rn) + 1

INC direct
Bytes: 2
Cycles: 1

Encoding: 0 0 0 0 0 1 0 1 direct address

Operation: INC (direct) ← (direct) + 1
INC @Ri

Bytes: 1
Cycles: 1

Encoding: 0 0 0 0 0 1 1 i

Operation: INC
((Ri) ← (Ri) + 1

INC DPTR

Function: Increment Data Pointer
Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^16) is performed; an overflow of the low-order byte of the data pointer (DPL) from OFFH to 00H will increment the high-order byte (DPH). No flags are affected.

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

INC DPTR
INC DPTR
INC DPTR

will change DPH and DPL to 13H and 01H.

Bytes: 1
Cycles: 2

Encoding: 1 0 1 0 0 0 1 1

Operation: INC
(DPTR) ← (DPTR) + 1
**MCS®-51 INSTRUCTION SET**

---

**JB**  
\( \text{bit,rel} \)

**Function:** Jump if Bit set  
**Description:** If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.  
**Example:** The data present at input port 1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence,  
\[
\begin{align*}
\text{JB} & \quad \text{P1.2,\text{LABEL1}} \\
\text{JB} & \quad \text{ACC.2,\text{LABEL2}}
\end{align*}
\]
will cause program execution to branch to the instruction at label LABEL2.  
**Bytes:** 3  
**Cycles:** 2

**Encoding:**  
\[
\begin{array}{cccc}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0
\end{array}
\]
bit address  
rel. address

**Operation:**  
\[
\begin{align*}
\text{JB} & \\
(\text{PC}) & \leftarrow (\text{PC}) + 3 \\
\text{IF} & \quad \text{(bit)} = 1 \\
\text{THEN} & \\
(\text{PC}) & \leftarrow (\text{PC}) + \text{rel}
\end{align*}
\]

---

**JBC**  
\( \text{bit,rel} \)

**Function:** Jump if Bit is set and Clear bit  
**Description:** If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.  
**Note:** When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.  
**Example:** The accumulator holds 56H (01010110B). The instruction sequence,  
\[
\begin{align*}
\text{JBC} & \quad \text{ACC.3,\text{LABEL1}} \\
\text{JBC} & \quad \text{ACC.2,\text{LABEL2}}
\end{align*}
\]
will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52H (01010010B).
MCS®-51 INSTRUCTION SET

Bytes: 3
Cycles: 2

Encoding: 0 0 0 1 0 0 0 0

Operation:

JBC
(PC) ← (PC) + 3
IF (bit) = 1
THEN
(bit) ← 0
(PC) ← (PC) + rel

JC rel

Function: Jump if Carry is set
Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

JC LABEL1
CPL C
JC LABEL2

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2

Encoding: 0 1 0 0 0 0 0 0

Operation:

JC
(PC) ← (PC) + 2
IF (C) = 1
THEN
(PC) ← (PC) + rel
JMP @A + DPTR

Function: Jump indirect
Description: Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^16): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP__TBL:

```
MOV DPTR, #JMP__TBL
JMP @A + DPTR
JMP__TBL: AJMP LABEL0
          AJMP LABEL1
          AJMP LABEL2
          AJMP LABEL3
```

If the accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1
Cycles: 2

Encoding: 0111 0011

Operation: JMP
(PC) ← (A) + (DPTR)
MCS®-51 INSTRUCTION SET

JNB bit,rel

Function: Jump if Bit Not set
Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Example: The data present at input port 1 is 11001010B. The accumulator holds 56H (01010110B). The instruction sequence,

```
JNB P1.3, LABEL1
JNB ACC.3, LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3
Cycles: 2

Encoding:

```
0 0 1 1 0 0 0 0
```

bit address rel. address

Operation:

```
JNB
(PC) ← (PC) + 3
IF (bit) = 0
THEN (PC) ← (PC) + rel.
```

JNC rel

Function: Jump if Carry not set
Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC LABEL1
CPL C
JNC LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.
MCS®-51 INSTRUCTION SET

JNZ rel

Function: Jump if accumulator Not Zero
Description: If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.
Example: The accumulator originally holds 00H. The instruction sequence,

```assembly
JNZ LABEL1
INC A
JNZ LABEL2
```

will set the accumulator to 01H and continue at label LABEL2.

Bytes: 2
Cycles: 2

Encoding: 0 1 0 1 0 0 0 0 rel. address

Operation: JNZ
(PC) ← (PC) + 2
IF (C) = 0
THEN (PC) ← (PC) + rel

Bytes: 2
Cycles: 2

Encoding: 0 1 1 1 0 0 0 0 rel. address

Operation: JNZ
(PC) ← (PC) + 2
IF (A) ≠ 0
THEN (PC) ← (PC) + rel

8-32
MCS®-51 INSTRUCTION SET

JZ rel

**Function:** Jump if Accumulator Zero  
**Description:** If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.  
**Example:** The accumulator originally contains 01H. The instruction sequence,  

```
JZ LABEL1  
DEC A  
JZ LABEL2
```

will change the accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.  
**Bytes:** 2  
**Cycles:** 2  
**Encoding:**  

```
0 1 1 0 0 0 0 0 0 0
```

**Operation:**  

```
JZ  
(PC)←(PC) + 2  
IF (A) = 0  
THEN (PC)←(PC) + rel
```

LCALL addr16

**Function:** Long Call  
**Description:** LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.
Example: Initially the stack pointer equals 07H. The label “SUBRTN” is assigned to program memory location 1234H. After executing the instruction,

```
LCALL SUBRTN
```

at location 0123H, the stack pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Encoding:**

```
0 0 0 1 0 0 1 0
```

**Operation:**

```
LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC7-0)
(SP) ← (SP) + 1
((SP)) ← (PC15-8)
(PC) ← addr15-0
```

---

**LJMP addr16**

**Function:** Long Jump

**Description:** LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

**Example:** The label “JMPADR” is assigned to the instruction at program memory location 1234H. The instruction,

```
LJMP JMPADR
```

at location 0123H will load the program counter with 1234H.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Encoding:**

```
0 0 0 0 0 0 1 0
```

**Operation:**

```
LJMP
(PC) ← addr15-0
```
**MCS®-51 INSTRUCTION SET**

**MOV** <dest-byte>,<src-byte>

<table>
<thead>
<tr>
<th>Function:</th>
<th>Move byte variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.</td>
</tr>
<tr>
<td></td>
<td>This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.</td>
</tr>
<tr>
<td>Example:</td>
<td>Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).</td>
</tr>
</tbody>
</table>

| MOV R0,#30H     | ;R0< = 30H                                              |
| MOV A,@R0       | ;A < = 40H                                              |
| MOV R1,A        | ;R1 < = 40H                                             |
| MOV R,@R1       | ;B < = 10H                                              |
| MOV @R1,P1      | ;RAM (40H) < = 0CAH                                     |
| MOV P2, P1      | ;P2 #0CAH                                               |

leaves the value 30H in register 0, 40H in both the accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

| MOV A,Rn        |                                             |
| Bytes:          | 1                                           |
| Cycles:         | 1                                           |
| Encoding:       | 11101rrr                                       |
| Operation:      | MOV (A) ← (Rn)                                |

**MOV A,direct**

| Bytes:          | 2                                           |
| Cycles:         | 1                                           |
| Encoding:       | 11100101 direct address                      |
| Operation:      | MOV (A) ← (direct)                           |

*MOV A,ACC is not a valid instruction.*
MCS®-51 INSTRUCTION SET

MOV A,@Ri
Bytes: 1
Cycles: 1
Encoding: 1 1 1 1 0 0 1 1 i
Operation: MOV (A) ← ((Ri))

MOV A,#data
Bytes: 2
Cycles: 1
Encoding: 0 1 1 1 0 1 0 0 immediate data
Operation: MOV (A) ← #data

MOV Rn,A
Bytes: 1
Cycles: 1
Encoding: 1 1 1 1 1 r r r
Operation: MOV (Rn) ← (A)

MOV Rn,direct
Bytes: 2
Cycles: 2
Encoding: 1 0 1 0 1 r r r direct addr.
Operation: MOV (Rn) ← (direct)

MOV Rn,#data
Bytes: 2
Cycles: 1
Encoding: 0 1 1 1 1 r r r immediate data
Operation: MOV (Rn) ← #data
MCS®-51 INSTRUCTION SET

MOV direct,A
Bytes: 2
Cycles: 1

Encoding: 11110101
Operation: MOV (direct) ← (A)

MOV direct,Rn
Bytes: 2
Cycles: 2

Encoding: 10001rrr
Operation: MOV (direct) ← (Rn)

MOV direct,direct
Bytes: 3
Cycles: 2

Encoding: 10000101
Operation: MOV (direct) ← (direct)

MOV direct,@Ri
Bytes: 2
Cycles: 2

Encoding: 1000011i
Operation: MOV (direct) ← ([Ri])

MOV direct,#data
Bytes: 3
Cycles: 2

Encoding: 01110101
Operation: MOV (direct) ← #data
### MOV @Ri,A

- **Bytes:** 1
- **Cycles:** 1
- **Encoding:** 1111 0111
- **Operation:** MOV \((\text{Ri}) \leftarrow (A)\)

### MOV @Ri,direct

- **Bytes:** 2
- **Cycles:** 2
- **Encoding:** 1010 0111 direct addr
- **Operation:** MOV \((\text{Ri}) \leftarrow \text{direct}\)

### MOV @Ri,#data

- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 0111 0111 immediate data
- **Operation:** MOV \((\text{Ri}) \leftarrow \#\text{data}\)

### MOV <dest-bit>,<src-bit>

<table>
<thead>
<tr>
<th>Function</th>
<th>Move bit data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.</td>
</tr>
<tr>
<td>Example</td>
<td>The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output port 1 is 35H (00110101B). MOV P1.3,C MOV C,P3.3 MOV P1.2,C will leave the carry cleared and change port 1 to 39H (00111001B).</td>
</tr>
</tbody>
</table>
MOV C, bit
Bytes: 2
Cycles: 1
Encoding: 1 0 1 0 0 0 1 0 bit address
Operation: MOV (C) ← (bit)

MOV bit, C
Bytes: 2
Cycles: 2
Encoding: 1 0 0 1 0 0 1 0 bit address
Operation: MOV (bit) ← (C)

MOV DPTR, #data16

Function: Load Data Pointer with a 16-bit constant
Description: The data pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

Example:
This is the only instruction which moves 16 bits of data at once.
The instruction,
MOV DPTR, #1234H
will load the value 1234H into the data pointer: DPH will hold 12H and DPL will hold 34H.
Bytes: 3
Cycles: 2
Encoding: 1 0 0 1 0 0 0 0 immed. data15 - 8 immed. data7 - 0
Operation: MOV (DPTR) ← #data15 - 0
DPH  DPL ← #data15 - 8  #data7 - 0
MOVC A, @A + <base-reg>

**Function:** Move Code byte

**Description:** The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the accumulator: otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

**Example:** A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC A
        MOVC A, @A + PC
        RET
        DB 66H
        DB 77H
        DB 88H
        DB 99H
```

If the subroutine is called with the accumulator equal to 01H, it will return with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table; the corresponding number would be added to the accumulator instead.

**MOVC A, @A + DPTR**

**Bytes:** 1

**Cycles:** 2

**Encoding:**

```
0011
```

**Operation:**

```
MOVC (A) ← ((A) + (DPTR))
```
MCS®-51 INSTRUCTION SET

MOVC A,@A + PC

- **Bytes:** 1
- **Cycles:** 2
- **Encoding:** 100000011

**Operation:**

- \((PC) \leftarrow (PC) + 1\)
- \((A) \leftarrow ((A) + (PC))\)

### MOVX <dest-byte>,<src-byte>

<table>
<thead>
<tr>
<th>Function:</th>
<th>Move External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the “X” appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.</td>
</tr>
</tbody>
</table>

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.
Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel® 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX A,@R1
MOVX @R0,A
```

copies the value 56H into both the accumulator and external RAM location 12H.

**MOVX A,@Ri**
- **Bytes:** 1
- **Cycles:** 2
- **Encoding:** 1110001i
- **Operation:** MOVX (A)←→((Ri))

**MOVX A,@DPTR**
- **Bytes:** 1
- **Cycles:** 2
- **Encoding:** 11100000
- **Operation:** MOVX (A)←→((DPTR))

**MOVX @Ri,A**
- **Bytes:** 1
- **Cycles:** 2
- **Encoding:** 1111001i
- **Operation:** MOVX ((Ri))←→(A)

**MOVX @DPTR,A**
- **Bytes:** 1
- **Cycles:** 2
- **Encoding:** 11110000
- **Operation:** MOVX (DPTR)←→(A)
MUL  AB

Function:  Multiply
Description:  MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.
Example:  Originally the accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,
          MUL  AB
          will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.
Bytes:  1
Cycles:  4
Encoding:  [1 0 1 0 0 1 0 0]
Operation:  MUL
            (A)7-0 ← (A) X (B)
            (B)15-8
NOP

Function: No Operation
Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.
Example: It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7

Bytes: 1
Cycles: 1
Encoding: 0 0 0 0 0 0 0 0
Operation: NOP
(PC) ← (PC) + 1
MCS®-51 INSTRUCTION SET

**ORL** `{dest-byte} {src-byte}`

**Function:** Logical-OR for byte variables

**Description:** ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** If the accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL A,R0
```

will leave the accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the accumulator at run-time. The instruction,

```
ORL P1,#00110010B
```

will set bits 5, 4, and 1 of output port 1.

**ORL A,Rn**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

```
0 1 0 0 1 r r r
```

**Operation:**

```
ORL (A) ← (A) ∨ (Rn)
```
ORL A, direct
Bytes: 2
Cycles: 1
Encoding: 0 1 0 0 0 1 0 1
direct address
Operation: ORL
(A) ← (A) ∨ (direct)

ORL A, @Ri
Bytes: 1
Cycles: 1
Encoding: 0 1 0 0 0 1 1
Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A, #data
Bytes: 2
Cycles: 1
Encoding: 0 1 0 0 0 1 0 0
immediate data
Operation: ORL
(A) ← (A) ∨ #data

ORL direct, A
Bytes: 2
Cycles: 1
Encoding: 0 1 0 0 0 0 1 0
direct address
Operation: ORL
(direct) ← (direct) ∨ (A)

ORL direct, #data
Bytes: 3
Cycles: 2
Encoding: 0 1 0 0 0 0 1 1
direct addr. immediate data
Operation: ORL
(direct) ← (direct) ∨ #data
MCS®-51 INSTRUCTION SET

ORL C, <src-bit>

**Function:** Logical-OR for bit variables

**Description:** Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

**Example:** Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:

```
MOV C,P1.0 ; LOAD CARRY WITH INPUT PIN P10
ORL C,ACC.7 ; OR CARRY WITH THE ACC. BIT 7
ORL C,/.OV ; OR CARRY WITH THE INVERSE OF OV
```

**ORL C,bit**

**Bytes:** 2

**Cycles:** 2

**Encoding:** 0 1 1 1 0 0 1 0  bit address

**Operation:** 

\[(C) \leftarrow (C) \lor (\text{bit})\]

**ORL C,/bit**

**Bytes:** 2

**Cycles:** 2

**Encoding:** 1 0 1 0 0 0 0 0  bit address

**Operation:** 

\[(C) \leftarrow (C) \lor (\text{bit})\]

**POP direct**

**Function:** Pop from stack.

**Description:** The contents of the internal RAM location addressed by the stack pointer is read, and the stack pointer is decremented by one. The value read is the transfer to the directly addressed byte indicated. No flags are affected.
Example: The stack pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

POP DPH
POP DPL

will leave the stack pointer equal to the value 30H and the data pointer set to 0123H. At this point the instruction,

POP SP

will leave the stack pointer set to 20H. Note that in this special case the stack pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2
Cycles: 2

Encoding: 1 1 0 1 0 0 0 0        direct address

Operation: POP
            (direct) ← ((SP))
            (SP) ← (SP) - 1

PUSH direct

Function: Push onto stack
Description: The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.
Example: On entering an interrupt routine the stack pointer contains 09H. The data pointer holds the value 0123H. The instruction sequence,

PUSH DPL
PUSH DPH

will leave the stack pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2
Cycles: 2

Encoding: 1 1 0 0 0 0 0 0        direct address

Operation: PUSH
            (SP) ← (SP) + 1
            (SP) ← (direct)
RET

Function: Return from subroutine
Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.
Example: The stack pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RET
will leave the stack pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1
Cycles: 2

Encoding: 0 0 1 0 0 0 1 0

Operation:
RET
(PC15:8) ← (SP)
(SP) ← (SP) - 1
(PC7:0) ← (SP)
(SP) ← (SP) - 1

RETI

Function: Return from interrupt
Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.
Example: The stack pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RETI

will leave the stack pointer equal to 09H and return program execution to location 0123H.

| Bytes: | 1 |
| Cycles: | 2 |

Encoding: 0 0 1 1 0 0 1 0

Operation:

RETI

(\text{PC}_{15-8}) \rightarrow ((\text{SP}))

(\text{SP}) \rightarrow (\text{SP}) - 1

(\text{PC}_{7-0}) \rightarrow ((\text{SP}))

(\text{SP}) \rightarrow (\text{SP}) - 1
**MCS®-51 INSTRUCTION SET**

### RL A

**Function:** Rotate accumulator Left

**Description:** The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

**Example:** The accumulator holds the value OC5H (11000101B). The instruction, 

```
RL A
```

leaves the accumulator holding the value 8BH (10001011B) with the carry unaffected.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

```
0 0 1 0 0 0 1 1
```

**Operation:**

```
RL
(An + 1) ← (An)  n = 0 - 6
(A0) ← (A7)
```

### RLC A

**Function:** Rotate accumulator Left through the Carry flag

**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

**Example:** The accumulator holds the value OC5H (11000101B), and the carry is zero. The instruction, 

```
RLC A
```

leaves the accumulator holding the value 8BH (10001010B) with the carry set.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

```
0 0 1 1 0 0 1 1
```

**Operation:**

```
RLC
(An + 1) ← (An)  n = 0 - 6
(A0) ← (C)
(C) ← (A7)
```

8-51
### RR A

**Function:** Rotate accumulator Right  
**Description:** The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.  
**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,  
\[ \text{RR } A \]  
leaves the accumulator holding the value 0E2H (11100010B) with the carry unaffected.  
**Bytes:** 1  
**Cycles:** 1  
**Encoding:**  
\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
\end{array}
\]  
**Operation:**  
\[ \text{RR} \]  
\[
(\text{An}) \leftarrow (\text{An} + 1) \quad n = 0 - 6 \\
(\text{A7}) \leftarrow (\text{A0}) \\
\]

### RRC A

**Function:** Rotate accumulator Right through Carry flag  
**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.  
**Example:** The accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,  
\[ \text{RRC } A \]  
leaves the accumulator holding the value 62 (01100010B) with the carry set.  
**Bytes:** 1  
**Cycles:** 1  
**Encoding:**  
\[
\begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
\end{array}
\]  
**Operation:**  
\[ \text{RRC} \]  
\[
(\text{An}) \leftarrow (\text{An} + 1) \quad n = 0 - 6 \\
(\text{A7}) \leftarrow (\text{C}) \\
(\text{C}) \leftarrow (\text{A0}) \\
\]
**SETB**  <bit>

<table>
<thead>
<tr>
<th>Function:</th>
<th>Set Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.</td>
</tr>
<tr>
<td>Example:</td>
<td>The carry flag is cleared. Output port 1 has been written with the value 34H (00110100B). The instructions,</td>
</tr>
</tbody>
</table>
| | SETB C  
| | SETB P1.0  
| | will leave the carry flag set to 1 and change the data output on port 1 to 35H (00110101B). |

**SETB**  C  

| Bytes: | 1 |
| Cycles: | 1 |
| Encoding: | 11010011 |
| Operation: | SETB  
| (C)← 1 |

**SETB**  bit  

| Bytes: | 2 |
| Cycles: | 1 |
| Encoding: | 11010010  
| bit address |
| Operation: | SETB  
| (bit)← 1 |

**SJMP**  rel  

| Function: | Short Jump |
| Description: | Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it. |
Example: The label “RELADR” is assigned to an instruction at program memory location 0123H. The instruction,

```
SJMP   RELADR
```

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

*(Note: Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)*

**Bytes:** 2  
**Cycles:** 2

**Encoding:**  
```
1 0 0 0 0 0 0 0 0 0
```
*rel. address*

**Operation:**  
```
(SJMP) ← (PC) + 2  
(PI) ← (PC) + rel
```
SUBB  A, <src-byte>

**Function:** Subtract with borrow

**Description:** SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds OC9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

```
SUBB   A,R2
```

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that OC9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

**SUBB  A,Rn**

| Bytes: | 1 |
| Cycles: | 1 |
| Encoding: | 10011rrr |
| Operation: | SUBB  
(A) ← (A) − (C) − (Rn) |
**SUBB A, direct**

- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** `1 0 0 1 0 1 0 1`
  - **Operation:** SUBB
    - `(A) ← (A) − (C) − (direct)`

**SUBB A, @Ri**

- **Bytes:** 1
- **Cycles:** 1
- **Encoding:** `1 0 0 1 0 1 1`
  - **Operation:** SUBB
    - `(A) ← (A) − (C) − ((Ri))`

**SUBB A, #data**

- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** `1 0 0 1 0 1 0 0`
  - **Operation:** SUBB
    - `(A) ← (A) − (C) − #data`

**SWAP A**

- **Function:** Swap nibbles within the Accumulator
- **Description:** SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.
- **Example:** The accumulator holds the value 0C5H (11000101B). The instruction,
  - `SWAP A`
leaves the accumulator holding the value 5CH (01011100B).
- **Bytes:** 1
- **Cycles:** 1
- **Encoding:** `1 1 0 0 0 1 0 0`
  - **Operation:** SWAP
    - `(A3-0) ← (A7-4), (A7-4) ← (A3-0)`
**MCS®-51 INSTRUCTION SET**

**XCH A,<byte>**

**Function:** Exchange Accumulator with byte variable

**Description:** XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

**Example:** R0 contains the address 20H. The accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

```
XCH A,@R0
```

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

**XCH A,Rn**

<table>
<thead>
<tr>
<th>Bytes:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles:</td>
<td>1</td>
</tr>
</tbody>
</table>

**Encoding:**

```
1100|rrrr
```

**Operation:**

```
XCH (A) ← (Rn)
```

**XCH A,direct**

<table>
<thead>
<tr>
<th>Bytes:</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles:</td>
<td>1</td>
</tr>
</tbody>
</table>

**Encoding:**

```
1100|0101[direct address]
```

**Operation:**

```
XCH (A) ← (direct)
```

**XCH A,@Ri**

<table>
<thead>
<tr>
<th>Bytes:</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles:</td>
<td>1</td>
</tr>
</tbody>
</table>

**Encoding:**

```
1100|0111i
```

**Operation:**

```
XCH (A) ← ((Ri))
```
MCS®-51 INSTRUCTION SET

XCHD  A, @Ri

Function: Exchange Digit
Description: XCHD exchanges the low-order nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD  A, @R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1
Cycles: 1

Encoding: 1 1 0 1 0 1 1 i

Operation: XCHD
(A3-0) \rightarrow ((Ri3-0))
### XRL <dest-byte>,<src-byte>

<table>
<thead>
<tr>
<th>Function:</th>
<th>Logical Exclusive-OR for byte variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.</td>
</tr>
</tbody>
</table>

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

(Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.)

<table>
<thead>
<tr>
<th>Example:</th>
<th>If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction, XRL A,R0 will leave the accumulator holding the value 69H (01101001B).</th>
</tr>
</thead>
</table>

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction, XRL P1,#00110001B will complement bits 5, 4, and 0 of output port 1.

<table>
<thead>
<tr>
<th>XRL A,Rn</th>
<th>Bytes: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles: 1</td>
</tr>
<tr>
<td>Encoding:</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>Operation:</td>
<td>XRL (A) ← (A) ⊕ (Rn)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XRL A,direct</th>
<th>Bytes: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles: 1</td>
</tr>
<tr>
<td>Encoding:</td>
<td>0 1 1 0 0 1 0 1</td>
</tr>
<tr>
<td>Operation:</td>
<td>XRL (A) ← (A) ⊕ (direct)</td>
</tr>
</tbody>
</table>
**MCS®-51 INSTRUCTION SET**

**XRL A,@Ri**
- Bytes: 1
- Cycles: 1
- Encoding: 0 1 1 0 0 1 1 1
- Operation: XRL
  \((A) \leftarrow (A) \oplus (\overline{\text{Ri}})\)

**XRL A,#data**
- Bytes: 2
- Cycles: 1
- Encoding: 0 1 1 0 0 1 0 0
  - immediate data
- Operation: XRL
  \((A) \leftarrow (A) \oplus \#\text{data}\)

**XRL direct,A**
- Bytes: 2
- Cycles: 1
- Encoding: 0 1 1 0 0 0 1 0
  - direct address
- Operation: XRL
  \((A) \leftarrow (\text{direct}) \oplus (A)\)

**XRL direct,#data**
- Bytes: 3
- Cycles: 2
- Encoding: 0 1 1 0 0 0 1 1
  - direct address
  - immediate data
- Operation: XRL
  \((\text{direct}) \leftarrow (\text{direct}) \oplus \#\text{data}\)
MCS®-51
8-BIT CONTROL-ORIENTED MICROCOMPUTERS

8031/8051
8031AH/8051AH
8032AH/8052AH
8751H/8751H-12

- High Performance HMOS Process
- Internal Timers/Event Counters
- 2-Level Interrupt Priority Structure
- 32 I/O Lines (Four 8-Bit Ports)
- 64K Program Memory Space

- Boolean Processor
- Bit-Addressable RAM
- Programmable Full Duplex Serial Channel
- 111 Instructions (64 Single-Cycle)
- 64K Data Memory Space

Security Feature Protects EPROM Parts Against Software Piracy

The MCS®-51 products are optimized for control applications. Byte-processing and numerical operations on small data structures are facilitated by a variety of fast addressing modes for accessing the internal RAM. The instruction set provides a convenient menu of 8-bit arithmetic instructions, including multiply and divide instructions. Extensive on-chip support is provided for one-bit variables as a separate data type, allowing direct bit manipulation and testing in control and logic systems that require Boolean processing.

<table>
<thead>
<tr>
<th>Device</th>
<th>Program</th>
<th>Data</th>
<th>Timers/Event Counters</th>
<th>Interrupts</th>
</tr>
</thead>
<tbody>
<tr>
<td>8052AH</td>
<td>8K x 8 ROM</td>
<td>256 x 8 RAM</td>
<td>3 x 16-Bit</td>
<td>6</td>
</tr>
<tr>
<td>8051AH</td>
<td>4K x 8 ROM</td>
<td>128 x 8 RAM</td>
<td>2 x 16-Bit</td>
<td>5</td>
</tr>
<tr>
<td>8051AH</td>
<td>4K x 8 ROM</td>
<td>128 x 8 RAM</td>
<td>2 x 16-Bit</td>
<td>5</td>
</tr>
<tr>
<td>8032AH</td>
<td>none</td>
<td>256 x 8 RAM</td>
<td>3 x 16-Bit</td>
<td>6</td>
</tr>
<tr>
<td>8031AH</td>
<td>none</td>
<td>128 x 8 RAM</td>
<td>2 x 16-Bit</td>
<td>5</td>
</tr>
<tr>
<td>8031AH</td>
<td>none</td>
<td>128 x 8 RAM</td>
<td>2 x 16-Bit</td>
<td>5</td>
</tr>
<tr>
<td>8751H</td>
<td>4K x 8 EPROM</td>
<td>128 x 8 RAM</td>
<td>2 x 16-Bit</td>
<td>5</td>
</tr>
<tr>
<td>8751H-12</td>
<td>4K x 8 EPROM</td>
<td>128 x 8 RAM</td>
<td>2 x 16-Bit</td>
<td>5</td>
</tr>
</tbody>
</table>

The 8751H is an EPROM version of the 8051AH; that is, the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. It is fully compatible with its predecessor, the 8751-8, but incorporates two new features: a Program Memory Security bit that can be used to protect the EPROM against unauthorized read-out, and a programmable baud rate modification bit (SMOD). SMOD is not present in the 8751H-12.
PIN DESCRIPTIONS

VCC
Supply voltage.

VSS
Circuit ground.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink 8 LS TTL inputs. Port 0 pins that have 1s written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1s, and can source and sink 8 LS TTL inputs.

Port 0 also receives the code bytes during programming of the EPROM parts, and outputs the code bytes during program verification of the ROM and EPROM parts. External pullups are required during program verification.

Port 1
Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source 4 LS TTL inputs. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during programming of the EPROM parts and during program verification of the ROM and EPROM parts.
In the 8032AH and 8052AH, Port 1 pins P1.0 and P1.1 also serve the T2 and T2EX functions, respectively.

**Port 2**

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL inputs. Port 2 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits during programming of the EPROM parts and during program verification of the ROM and EPROM parts.

**Port 3**

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source 4 LS TTL inputs. Port 3 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.

Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternative Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RXD (serial input port)</td>
</tr>
<tr>
<td>P3.1</td>
<td>TXD (serial output port)</td>
</tr>
<tr>
<td>P3.2</td>
<td>INT0 (external interrupt 0)</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 (external interrupt 1)</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 (Timer 0 external input)</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 (Timer 1 external input)</td>
</tr>
<tr>
<td>P3.6</td>
<td>WR (external data memory write strobe)</td>
</tr>
<tr>
<td>P3.7</td>
<td>RD (external data memory read strobe)</td>
</tr>
</tbody>
</table>
RST
Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/PROG
Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE can drive 8 LS TTL inputs. This pin is also the program pulse input (PROG) during programming of the EPROM parts.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

PSEN
Program Store Enable is the read strobe to external Program Memory. PSEN can drive 8 LS TTL inputs.

When the device is executing code from external Program Memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external Data Memory.

EA/VPP
External Access enable EA must be externally held low in order to enable any MCS-51 device to fetch code from external Program Memory locations 0 to 0FFFH (0 to 1FFFH, in the 8032AH and 8052AH).

Note, however, that if the Security Bit in the EPROM devices is programmed, the device will not fetch code from any location in external Program Memory.

This pin also receives the 21V programming supply voltage (VPP) during programming of the EPROM parts.

XTAL1
Input to the inverting oscillator amplifier.

XTAL2
Output from the inverting oscillator amplifier.

OSCILLATOR CHARACTERISTICS
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 3. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, "Oscillators for Microcontrollers."

To drive the device from an external clock source, XTAL1 should be grounded, while XTAL2 is driven, as shown in Figure 4. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

Figure 3. Oscillator Connections

Figure 4. External Drive Configuration
**ABSOLUTE MAXIMUM RATINGS**

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

Ambient Temperature Under Bias . . . .0 °C to 70 °C  
Storage Temperature . . . . . . -65 °C to +150 °C  
Voltage on EA/VPP Pin to VSS . . . . -0.5V to +21.5V  
Voltage on Any Other Pin to VSS . . . . -0.5V to +7V  
Power Dissipation  . . . . . . . . . . 1.5W

---

### D.C. CHARACTERISTICS: (T_A = 0 °C to 70 °C; VCC = 5V ±10%; VSS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage (Except EA Pin of 8751H, 8751H-12)</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIL1</td>
<td>Input Low Voltage to EA Pin of 8751H, 8751H-12</td>
<td>0</td>
<td>0.7</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (Except XTAL2, RST)</td>
<td>2.0</td>
<td>VCC +0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage to XTAL2, RST</td>
<td>2.5</td>
<td>VCC +0.5</td>
<td>V</td>
<td>XTAL1 = VSS</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage (Ports 1, 2, 3)*</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>IOL = 1.6 mA</td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage (Port 0, ALE, PSEN)*</td>
<td>0.60</td>
<td></td>
<td>V</td>
<td>IOL = 3.2 mA</td>
</tr>
<tr>
<td></td>
<td>8751H, 8751H-12</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>IOL = 2.4 mA</td>
</tr>
<tr>
<td></td>
<td>All Others</td>
<td>0.45</td>
<td></td>
<td>V</td>
<td>IOL = 3.2 mA</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage (Ports 1, 2, 3)</td>
<td>2.4</td>
<td></td>
<td>V</td>
<td>IOH = -80 μA</td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN)</td>
<td>2.4</td>
<td></td>
<td>V</td>
<td>IOH = -400 μA</td>
</tr>
<tr>
<td>IIL</td>
<td>Logical 0 Input Current (Ports 1, 2, 3)</td>
<td>-800</td>
<td></td>
<td>μA</td>
<td>Vin = 0.45 V</td>
</tr>
<tr>
<td></td>
<td>8032AH, 8052AH</td>
<td>-500</td>
<td></td>
<td>μA</td>
<td>Vin = 0.45 V</td>
</tr>
<tr>
<td></td>
<td>All Others</td>
<td>-50</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>IIL1</td>
<td>Logical 0 Input Current to EA Pin of 8751H, 8751H-12 Only</td>
<td>-15</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>IIL2</td>
<td>Logical 0 Input Current (XTAL2)</td>
<td>-3.2</td>
<td></td>
<td>mA</td>
<td>Vin = 0.45 V</td>
</tr>
<tr>
<td>ILI</td>
<td>Input Leakage Current (Port 0)</td>
<td>±100</td>
<td></td>
<td>μA</td>
<td>0.45 &lt; Vin &lt; VCC</td>
</tr>
<tr>
<td></td>
<td>8751H, 8751H-12</td>
<td>±10</td>
<td></td>
<td>μA</td>
<td>0.45 &lt; Vin &lt; VCC</td>
</tr>
<tr>
<td></td>
<td>All Others</td>
<td>±10</td>
<td></td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td>IIH</td>
<td>Logical 1 Input Current to EA Pin of 8751H, 8751H-12</td>
<td>500</td>
<td></td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td>IIH1</td>
<td>Input Current to RST to Activate Reset</td>
<td>500</td>
<td></td>
<td>μA</td>
<td>Vin &lt; (VCC - 1.5V)</td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current: 8031/8051</td>
<td>160</td>
<td></td>
<td>mA</td>
<td>All Outputs Disconnected; EA = VCC</td>
</tr>
<tr>
<td></td>
<td>8031AH/8051AH</td>
<td>125</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8032AH/8052AH</td>
<td>175</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8751H/8751H-12</td>
<td>250</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>CIO</td>
<td>Pin Capacitance</td>
<td>10</td>
<td></td>
<td>pF</td>
<td>test freq = 1MHz</td>
</tr>
</tbody>
</table>

*Note: Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLs of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading > 100 pF), the noise pulse on the ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.*

---

9-5
### A.C. CHARACTERISTICS:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12MHz Osc</th>
<th>Variable Oscillator</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/TCLCL</td>
<td>Oscillator Frequency</td>
<td></td>
<td></td>
<td>MHz</td>
</tr>
<tr>
<td>TLHLL</td>
<td>ALE Pulse Width</td>
<td>127</td>
<td>2TCLCL-40</td>
<td>ns</td>
</tr>
<tr>
<td>TAVLL</td>
<td>Address Valid to ALE Low</td>
<td>43</td>
<td>TCLCL-40</td>
<td>ns</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold After ALE Low</td>
<td>48</td>
<td>TCLCL-35</td>
<td>ns</td>
</tr>
<tr>
<td>TLLIV</td>
<td>ALE Low to Valid Instr In 8751H, 8751H-12 All Others</td>
<td>183</td>
<td>4TCLCL-150</td>
<td>ns</td>
</tr>
<tr>
<td>TLLPL</td>
<td>ALE Low to PSEN Low</td>
<td>58</td>
<td>TCLCL-25</td>
<td>ns</td>
</tr>
<tr>
<td>TPLPH</td>
<td>PSEN Pulse Width 8751H, 8751H-12 All Others</td>
<td>190</td>
<td>3TCLCL-60</td>
<td>ns</td>
</tr>
<tr>
<td>TPLV</td>
<td>PSEN Low to Valid Instr In 8751H, 8751H-12 All Others</td>
<td>100</td>
<td>3TCLCL-150</td>
<td>ns</td>
</tr>
<tr>
<td>TXP</td>
<td>Input Instr Hold After PSEN</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TXIZ</td>
<td>Input Instr Float After PSEN</td>
<td>63</td>
<td>TCLCL-20</td>
<td>ns</td>
</tr>
<tr>
<td>TXAV</td>
<td>PSEN to Address Valid</td>
<td>75</td>
<td>TCLCL-8</td>
<td>ns</td>
</tr>
<tr>
<td>TAV</td>
<td>Address to Valid Instr In 8751H, 8751H-12 All Others</td>
<td>267</td>
<td>5TCLCL-150</td>
<td>ns</td>
</tr>
<tr>
<td>TPLAZ</td>
<td>PSEN Low to Address Float</td>
<td></td>
<td>TBD</td>
<td>ns</td>
</tr>
<tr>
<td>TRLRH</td>
<td>RD Pulse Width</td>
<td>400</td>
<td>6TCLCL-100</td>
<td>ns</td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR Pulse Width</td>
<td>400</td>
<td>6TCLCL-100</td>
<td>ns</td>
</tr>
<tr>
<td>TRLDV</td>
<td>RD Low to Valiad Data In</td>
<td>252</td>
<td>5TCLCL-165</td>
<td>ns</td>
</tr>
<tr>
<td>TRHDX</td>
<td>Data Hold After RD</td>
<td>0</td>
<td>2TCLCL-70</td>
<td>ns</td>
</tr>
<tr>
<td>TRHZD</td>
<td>Data Float After RD</td>
<td>97</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TLLDV</td>
<td>ALE Low to Valid Data In</td>
<td>517</td>
<td>8TCLCL-150</td>
<td>ns</td>
</tr>
<tr>
<td>Tavad</td>
<td>Address to Valid Data In</td>
<td>585</td>
<td>9TCLCL-165</td>
<td>ns</td>
</tr>
<tr>
<td>TLLWL</td>
<td>ALE Low to RD or WR Low</td>
<td>200</td>
<td>3TCLCL-50</td>
<td>ns</td>
</tr>
<tr>
<td>TAWWL</td>
<td>Address to RD or WR Low</td>
<td>203</td>
<td>4TCLCL-130</td>
<td>ns</td>
</tr>
<tr>
<td>TQVWX</td>
<td>Data Valid to WR Transition 8751H, 8751H-12 All Others</td>
<td>13</td>
<td>TCLCL-70</td>
<td>ns</td>
</tr>
<tr>
<td>TQVWH</td>
<td>Data Valid to WR High</td>
<td>433</td>
<td>7TCLCL-150</td>
<td>ns</td>
</tr>
<tr>
<td>TWHQX</td>
<td>Data Held After WR</td>
<td>33</td>
<td>TCLCL-50</td>
<td>ns</td>
</tr>
<tr>
<td>TRLAZ</td>
<td>RD Low to Address Float</td>
<td></td>
<td>TBD</td>
<td>ns</td>
</tr>
<tr>
<td>TWHLH</td>
<td>RD or WR High to ALE High 8751H, 8751H-12 All Others</td>
<td>33</td>
<td>TCLCL-50</td>
<td>ns</td>
</tr>
<tr>
<td>9-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EXTERNAL PROGRAM MEMORY READ CYCLE

- TLHLL
- TAVLL
- TLLPL
- TPLPH
- TLLIV
- TPLIV
- TPLAZ
- TLLAX
- INSTR
- A0-A7
- A0-A7
- TAVIV
- A8-A15
- A8-A15
EXTERNAL DATA MEMORY READ CYCLE

- **ALE**
  - TLHLL
  - TWHLH

- **PSEN**
  - TLLDV
  - TLLWL
  - TRLRH

- **RD**
  - TRLDV
  - TRLAZ
  - TRHDLZ

- **PORT 0**
  - A0–A7 FROM RI OR DPL
  - TLLAX
  - DATA IN
  - INSTR IN
  - A0–A7 FROM PCL
  - TAVWS
  - TAVDV

- **PORT 2**
  - P2.0–P2.7 OR A8–A15 FROM DPH
  - A8–A15 FROM PCH
EXTERNAL DATA MEMORY WRITE CYCLE

ALE

PSEN

WR

PORT 0

PORT 2

DATA OUT

A0–A7 FROM PCL

INSTR IN

A0–A7 FROM RI OR DPL

TAVLL

TLLLX

TQVWX

TQVWH

TWHQX

TLLWL

TWLWH

INTERNAL DATA MEMORY WRITE CYCLE
SERIAL PORT TIMING — SHIFT REGISTER MODE
Test Conditions: T_A = 0 °C to 70 °C; VCC = 5V ± 10%; VSS = 0V; Load Capacitance = 80 pF

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12MHz Osc</th>
<th>Variable Oscillator</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXLXL</td>
<td>Serial Port Clock Cycle Time</td>
<td>1.0</td>
<td>12TCLCL</td>
<td>µs</td>
</tr>
<tr>
<td>TQVXH</td>
<td>Output Data Setup to Clock Rising Edge</td>
<td>700</td>
<td>10TCLCL-133</td>
<td>ns</td>
</tr>
<tr>
<td>TXHQX</td>
<td>Output Data Hold After Clock Rising Edge</td>
<td>50</td>
<td>2TCLCL-117</td>
<td>ns</td>
</tr>
<tr>
<td>TXHDX</td>
<td>Input Data Hold After Clock Rising Edge</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TXHDV</td>
<td>Clock Rising Edge to Input Data Valid</td>
<td>700</td>
<td>10TCLCL-133</td>
<td>ns</td>
</tr>
</tbody>
</table>

SHIFT REGISTER TIMING WAVEFORMS
### EXTERNAL CLOCK DRIVE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/TCLCL</td>
<td>Oscillator Frequency</td>
<td>3.5</td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td>TCHCX</td>
<td>High Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCLCX</td>
<td>Low Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCLCH</td>
<td>Rise Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCHCL</td>
<td>Fall Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

### EXTERNAL CLOCK DRIVE WAVEFORMS

![Waveform Diagram]

### A.C. TESTING INPUT, OUTPUT WAVEFORM

![Waveform Diagram]

A.C. TESTING INPUTS ARE DRIVEN AT 2.4 V FOR A LOGIC 1 AND 0.45 V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 2.0 V FOR A LOGIC 1 AND 0.8 V FOR A LOGIC 0.
EPROM CHARACTERISTICS:

### Table 3. EPROM Programming Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>RST</th>
<th>PSEN</th>
<th>ALE</th>
<th>EA</th>
<th>P2.7</th>
<th>P2.6</th>
<th>P2.5</th>
<th>P2.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>1</td>
<td>0</td>
<td>0*</td>
<td>VPP</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Inhibit</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Verify</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Note: *1* = logic high for that pin
*0* = logic low for that pin
*"X"* = "don't care"
*"VPP"* = +21V ±0.5V

**Programming the EPROM**

To be programmed, the part must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the code byte to be programmed into that location is applied to Port 0. The other Port 2 pins, and RST, PSEN, and EA should be held at the "Program" levels indicated in Table 3. ALE is pulsed low for 50 msec to program the code byte into the addressed EPROM location. The setup is shown in Figure 5.

Normally EA is held at a logic high until just before ALE is to be pulsed. Then EA is raised to +21V, ALE is pulsed, and then EA is returned to a logic high. Waveforms and detailed timing specifications are shown in later sections of this data sheet.

Note that the EA/VPP pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

**Program Verification**

If the Security Bit has not been programmed, the on-chip Program Memory can be read out for verification purposes, if desired, either during or after the programming operation. The address of the Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3. The other pins should be held at the “Verify” levels indicated in Table 3. The contents of the addressed location will come out on Port 0. External pullups are required on Port 0 for this operation.

The setup, which is shown in Figure 6, is the same as for programming the EPROM except that pin P2.7 is held at a logic low, or may be used as an active-low read strobe.
**EPROM Security**

The security feature consists of a "locking" bit which when programmed denies electrical access by any external means to the on-chip Program Memory. The bit is programmed as shown in Figure 7. The setup and procedure are the same as for normal EPROM programming, except that P2.6 is held at a logic high. Port 0, Port 1, and pins P2.0–P2.3 may be in any state. The other pins should be held at the "Security" levels indicated in Table 3.

Once the Security Bit has been programmed, it can be cleared only by full erasure of the Program Memory. While it is programmed, the internal Program Memory can not be read out, the device can not be further programmed, and it can not execute out of external program memory. Erasing the EPROM, thus clearing the Security Bit, restores the device's full functionality. It can then be reprogrammed.

**Erasure Characteristics**

Erasure of the EPROM begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Angstroms. Since sunlight and flourescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level flourescent lighting) could cause inadvertent erasure. If an application subjects the device to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W-sec/cm². Exposing the EPROM to an ultraviolet lamp of 12,000 µW/cm² rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

### EPROM PROGRAMMING AND VERIFICATION CHARACTERISTICS:

(Tₐ = 21 °C to 27 °C, VCC = 5V±10%, VSS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPP</td>
<td>Programming Supply Voltage</td>
<td>20.5</td>
<td>21.5</td>
<td>V</td>
</tr>
<tr>
<td>IPP</td>
<td>Programming Supply Current</td>
<td>30</td>
<td></td>
<td>mA</td>
</tr>
<tr>
<td>1/TCLCL</td>
<td>Oscillator Frequency</td>
<td>4</td>
<td>6</td>
<td>MHz</td>
</tr>
<tr>
<td>TAVGL</td>
<td>Address Setup to PROG Low</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGHAX</td>
<td>Address Hold After PROG</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDVGL</td>
<td>Data Setup to PROG Low</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGHDX</td>
<td>Data Hold After PROG</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEHSH</td>
<td>P2.7 (ENABLE) High to VPP</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSHGL</td>
<td>VPP Setup to PROG Low</td>
<td>10</td>
<td></td>
<td>µsec</td>
</tr>
<tr>
<td>TGHSXL</td>
<td>VPP Hold After PROG</td>
<td>10</td>
<td></td>
<td>µsec</td>
</tr>
<tr>
<td>TGLGH</td>
<td>PROG Width</td>
<td>45</td>
<td>55</td>
<td>msec</td>
</tr>
<tr>
<td>TAVQV</td>
<td>Address to Data Valid</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TELQV</td>
<td>ENABLE Low to Data Valid</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEHQZ</td>
<td>Data Float After ENABLE</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EPROM PROGRAMMING AND VERIFICATION WAVEFORMS

PROGRAMMING

ADDRESS

DATA IN

TAVGQ

TDVGL

TAVGL

TSHGL

TGLGH 21V ± .5V

TGHSL

TTL HIGH

EA/VPP

TTL HIGH

P2.7

P1.0–P1.7

P2.0–P2.3

PORT 0

ADDRESS

DATA OUT

TAVQV

TGHDX

TGHAX

TTL HIGH

FOR PROGRAMMING CONDITIONS SEE FIGURE 5.

FOR VERIFICATION CONDITIONS SEE FIGURE 6.
8052AH-BASIC is an 8052AH microcontroller with a complete full-featured BASIC interpreter, MCS® BASIC-52, resident in the 8K of available ROM. This Software-On-Silicon product is specifically designed to address the needs of process control, measurement, and instrumentation applications. MCS BASIC-52 allows 8052AH users to write programs in the popular BASIC language, which is much simpler to write and easier to understand than assembly language.

In addition to the standard BASIC commands and functions, such as floating point arithmetic and transcendental operations, MCS BASIC-52 contains many unique features that allow the user to perform tasks that usually require assembly language. Bit-wise logical operators, such as AND, OR, and EXCLUSIVE-OR are supported as well as hexadecimal arithmetic.

A minimum amount of hardware is required to support MCS BASIC-52. Small systems can be constructed with only a latch, 1K bytes of external memory, and the appropriate serial port drivers. With the addition of a transistor, a gate, and a couple of passive components, MCS BASIC-52 can program EPROM or E2PROM devices with the users application program. Both the standard and the intelligent Programming™ algorithms are supported.

MCS BASIC-52 is an interpreted language. This allows the user to develop a program interactively without the cumbersome and repetitive process of editing, assembling, loading, and running which is required by assemblers and compilers. MCS BASIC-52 was designed to permit the programmer to develop resident high level language software using the high performance 8052AH device.
FEATURES

COMMAND SET
MCS BASIC-52 contains all standard BASIC commands, statements, and operators. Figure 1 lists the software feature set of MCS BASIC-52.

DATA FORMAT
The range of numbers that can be represented in MCS BASIC-52 is:

±1E−127 to ±.99999999E + 127

CONTROL ORIENTED FEATURES
MCS BASIC-52 contains many unique features to perform tasks that usually require assembly language programming. The XBY and DBY operators can read and/or write external and internal memory respectively. The CBY operator is used to read program memory. Additionally, virtually all of the special function registers on the 8052AH can be accessed with MCS BASIC-52. This allows the user to set the timer or interrupt modes within the constructs of a BASIC program. An accurate interrupt driven REAL TIME CLOCK that has a 5 millisecond resolution is also implemented in MCS BASIC-52. This clock can be enabled, disabled, and used to generate interrupts. Finally, a CALL statement that allows the programmer to CALL assembly language routines is available in MCS BASIC-52. Parameters can be passed in a number of different ways.

EPROM/E2PROM FILE
Most Basic interpreters allow only one program to be resident in memory, and many require that the program reside in RAM. MCS BASIC-52 allows programs to reside in both RAM and EPROM/E2PROM. Additionally, up to 255 programs may reside in EPROM/E2PROM. Programs may also be transferred (XFER) from EPROM/E2PROM to RAM for editing purposes.

EPROM/E2PROM PROGRAMMING
A powerful feature of MCS BASIC-52 is that it generates all of the timing necessary to program any standard EPROM or E2PROM device with the users’ program (PROG/FPROG). Additionally, very little external hardware is required to implement this feature. Saving programs in EPROM/E2PROM is much more attractive and reliable than other alternatives, such as cassette tape, especially in control and/or other noisy environments.

AUTOSTART
After the user programs an EPROM or E2PROM with the desired BASIC program. The PROG2 or FPROG2 commands may be used to enable the unique AUTOSTART feature of MCS BASIC-52. When AUTOSTART is enabled, MCS BASIC-52 will execute the user program after RESET or a power-up condition. This permits the user to RUN a program without connecting the MCS BASIC-52 device to a console — a powerful feature for control environments.

USER ACCESSIBLE FUNCTION LIBRARY
Another unique feature of MCS BASIC-52 is that it contains a complete library of functions that can be accessed with assembly language. All floating point, radix conversion, and I/O routines contained in MCS BASIC-52 can be accessed with assembly language CALL instructions. These complex arithmetic routines can be used by the programmer in applications requiring the speed of assembly language, but also the complex arithmetics offered by BASIC.

8052AH-BASIC PIN DESCRIPTION (FIGURE 2)
8052AH-BASIC is an 8052AH device, however, MCS BASIC-52 assumes a particular hardware configuration. The following pin description outlines the pin functions defined by MCS BASIC-52.

VSS
Circuit ground potential.

VCC
Circuit supply voltage. 5 volts ± 10% relative to VSS.

AD0–AD7
The multiplexed low-order address and data bus used during accesses to external memory. External pullup devices (~10K Ω) are required on these pins if the MCS BASIC-52 EPROM/E2PROM programming feature is used.

A8–A15
The high order address bus used during accesses to external memory.
commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Statements</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>BAUD</td>
<td>ADD (+)</td>
</tr>
<tr>
<td>LIST</td>
<td>CALL</td>
<td>DIVIDE (I)</td>
</tr>
<tr>
<td>LIST#</td>
<td>CLEAR</td>
<td>EXPONENTIATION (**)</td>
</tr>
<tr>
<td>NEW</td>
<td>CLEARS</td>
<td>MULTIPLY (*)</td>
</tr>
<tr>
<td>NULL</td>
<td>CLEARI</td>
<td>SUBTRACT (-)</td>
</tr>
<tr>
<td>RAM</td>
<td>CLOCK0</td>
<td>LOGICAL AND (AND)</td>
</tr>
<tr>
<td>ROM</td>
<td>CLOCK1</td>
<td>LOGICAL OR (OR)</td>
</tr>
<tr>
<td>XFERENCE</td>
<td>DATA</td>
<td>LOGICAL X-OR (.XOR)</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>READ</td>
<td>LOGICAL NOT</td>
</tr>
<tr>
<td>PROGRAM1</td>
<td>RESTORE</td>
<td>ABS ( )</td>
</tr>
<tr>
<td>PROGRAM2</td>
<td>DIM</td>
<td>INT ( )</td>
</tr>
<tr>
<td>FPROG</td>
<td>DO-WHILE</td>
<td>SGN ( )</td>
</tr>
<tr>
<td>FPROG1</td>
<td>DO-UNTIL</td>
<td>SQR ( )</td>
</tr>
<tr>
<td>FPROG2</td>
<td>END</td>
<td>RND</td>
</tr>
<tr>
<td>FOR-TO-STEP</td>
<td>LOG ( )</td>
<td></td>
</tr>
<tr>
<td>NEXT</td>
<td>EXP ( )</td>
<td></td>
</tr>
<tr>
<td>GOSUB</td>
<td>SIN ( )</td>
<td></td>
</tr>
<tr>
<td>RETURN</td>
<td>COS ( )</td>
<td></td>
</tr>
<tr>
<td>GOTO</td>
<td>TAN ( )</td>
<td></td>
</tr>
<tr>
<td>ON-GOTO</td>
<td>ATN ( )</td>
<td></td>
</tr>
<tr>
<td>ON-GOSUB</td>
<td>=, &lt;, &gt;, &gt;=</td>
<td></td>
</tr>
<tr>
<td>IF-THEN-ELSE</td>
<td>ASC ( )</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>CHR ( )</td>
<td></td>
</tr>
<tr>
<td>LET</td>
<td>CB ( )</td>
<td></td>
</tr>
<tr>
<td>ONERR</td>
<td>DBY ( )</td>
<td></td>
</tr>
<tr>
<td>ONEXIT1</td>
<td>XBY ( )</td>
<td></td>
</tr>
<tr>
<td>ONSERSE</td>
<td>GET</td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>IE</td>
<td></td>
</tr>
<tr>
<td>PRINT#</td>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>PH0</td>
<td>PORT1</td>
<td></td>
</tr>
<tr>
<td>PH0.</td>
<td>PCON</td>
<td></td>
</tr>
<tr>
<td>PH1</td>
<td>RCAP2</td>
<td></td>
</tr>
<tr>
<td>PH1.</td>
<td>T2CN</td>
<td></td>
</tr>
<tr>
<td>PUSH</td>
<td>TCON</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>TMOD</td>
<td></td>
</tr>
<tr>
<td>PWM</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>REM</td>
<td>TIMERO</td>
<td></td>
</tr>
<tr>
<td>RET1</td>
<td>TIMER1</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>TIMER2</td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>UI0</td>
<td>XTAL</td>
<td></td>
</tr>
<tr>
<td>UI1</td>
<td>MTOPO</td>
<td></td>
</tr>
<tr>
<td>UO0</td>
<td>LEN</td>
<td></td>
</tr>
<tr>
<td>UO1</td>
<td>FREE</td>
<td></td>
</tr>
<tr>
<td>PI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1. MCS® BASIC-52 Software Feature Set

PORT 1

A general purpose quasi-bidirectional 8-bit input/output port. The individual pins on PORT 1 all have alternate functions which may or may not be implemented by the user. The alternate functions are as follows:

PORT 1.0 (T2)

Can be used as the trigger input to TIMER/COUNTER 2. A one (1) must be written to this port pin output latch in order for this function to operate. Details of the T2 trigger function are covered in the Microcontrollers Handbook. Order Number 210918-002.

PORT 1.1 (T2EX)

Can be used as the external input to TIMER/COUNTER 2. A one (1) must be written to this port pin output latch in order for this function to operate. Details of the T2 trigger function are covered in the Microcontroller Users Manual.

PORT 1.2 (PWM OUTPUT)

This pin is used as the PWM output port when the PWM statement is executed. PWM stands for Pulse Width Modulation and is used to generate pulses of varying duty cycle and frequency.

PORT 1.3 (ALE DISABLE)

This pin is used to disable the ALE signal to the external address latch when the EPROM/E2PROM programming feature is used. In a system, this pin is logically anded with ALE.

PORT 1.4 (PROGRAMMING PULSE)

When the EPROM/E2PROM programming feature is used, this pin provides the proper programming pulse width to program EPROM and INTELligent EPROM® devices. MCS BASIC-52 actually calculates the proper programming pulse width from the system crystal value (XTAL) to assure the proper timing of this pulse. When used to program E2PROM devices, the length of this pulse is not critical. This pin is active in the logical zero (0) state.
PORT 1.5 (PROGRAMMING ENABLE)

When the EPROM/E2PROM programming feature is implemented, this pin is used to enable the EPROM programming voltage. This pin remains active (logically low (0)) during the entire EPROM programming process. On E2PROM devices that do not require any special programming voltage, this pin is not used.

PORT 1.6 (DMA ACKNOWLEDGE)

When the DMA feature is implemented as described in the MCS® BASIC-52 users manual, this pin functions as an active low DMA ACKNOWLEDGE output.

PORT 1.7 (LINE PRINTER OUTPUT)

This pin functions as a serial output port when the LIST# or PRINT# command and/or statement is used. This enables the user to make a “hard copy” of a program or to print out results of a calculation.

RESET

A high (2.5 volts) on this pin for two machine cycles while the oscillator is running resets the device. An external pulldown resistor (~8.2K) from RESET to VSS permits power-on reset when a capacitor (~10 uf) is connected from this pin to VCC.

ALE

ALE (address latch enable) is an output pin that is used to latch the low order address byte during Read, Write, or program fetch operations to external memory.

PSEN

This pin (Program Store ENable) is a control signal that is used to enable external program memory. In MCS® BASIC-52, this pin will always remain inactive (logically high (1)) unless the user is running an assembly language program in external memory.

XTAL1

Input to the inverting amplifier that forms the oscillator.

XTAL2

Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

RD

A control signal that is used to enable READ operations to external data memory. This pin is active low (0).

WR

A control signal that is used to enable WRITE operations to external data memory. This pin is active low (0).

T1

This pin can be programmed to be an external input to TIMER/COUNTER 1.

T0

This pin can be programmed to be an external input to TIMER/COUNTER 0.

INT1

This pin is the external interrupt 1 pin. It is active low and interrupts on this pin may be handled in either BASIC or in assembly language.

INT0/DMA REQUEST

This is the external interrupt 0 pin. It is active low and may be optionally programmed to function as a DMA request input pin. The DMA REQUEST pin is used by E2PROM devices during programming.

CONSOLE SERIAL OUTPUT

This is the serial output pin to the console device. Standard ASCII codes are used as well as a standard asynchronous frame.

CONSOLE SERIAL INPUT

This is the serial input pin that receives data from the console device. Standard ASCII codes are assumed to be the input and the data is assumed to be transmitted using a standard asynchronous frame.

NOTES

If pin 31 is grounded the '8052AH-BASIC will operate as a standard 8032AH. The tolerances on this pin are described under DC characteristics.

For detailed information concerning this product please refer to the MCS BASIC-52 Users Manual (Order Number 210918-002).
**ABSOLUTE MAXIMUM RATINGS**

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

Ambient Temperature Under Bias . . . 0°C to 70°C
Storage Temperature . . . . . . −65°C to +150°C
Voltage on Any Pin With Respect to Ground (VSS) . . . . . . −0.5V to +7V
Power Dissipation . . . . . . . . . . . . . . . 2 Watts

### DC CHARACTERISTICS (TA = 0°C to 70°C, VCC = 4.5V to 5.5V, VSS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>−0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (Except RST and XTAL2)</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage to RST for Reset, XTAL2</td>
<td>2.5</td>
<td>VCC + 0.5</td>
<td>V</td>
<td>XTAL1 to VSS</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage Port 1, A8–15, Control Functions</td>
<td>0.45</td>
<td>.V</td>
<td>IOL = 1.6mA</td>
<td></td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage ALE, PSEN (Note 1)</td>
<td>0.45</td>
<td>.V</td>
<td>IOL = 3.2mA</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage Port 1, A8–15, Control Functions</td>
<td>2.4</td>
<td>.V</td>
<td>IOH = −80μA</td>
<td></td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage ADO–7, ALE, PSEN</td>
<td>2.4</td>
<td>.V</td>
<td>IOH = −400μA</td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Logical 0 Input Current Port 1, A8–15 Control Functions</td>
<td>−800</td>
<td>μA</td>
<td>Vin = 0.45V</td>
<td></td>
</tr>
<tr>
<td>IIL2</td>
<td>Logical 0 Input Current XTAL2</td>
<td>−2.5</td>
<td>mA</td>
<td>XTAL1 at VSS, Vin = 0.45V</td>
<td></td>
</tr>
<tr>
<td>IL1</td>
<td>Input Leakage Current To ADO–7 EA</td>
<td>±10</td>
<td>μA</td>
<td>0.45V &lt; Vin &lt; VCC</td>
<td></td>
</tr>
<tr>
<td>IIH1</td>
<td>Input High Current to RST/VPD For Reset</td>
<td>500</td>
<td>μA</td>
<td>Vin = VCC − 1.5V</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current</td>
<td>175</td>
<td>mA</td>
<td>All outputs disconnected</td>
<td></td>
</tr>
<tr>
<td>CIO</td>
<td>Capacitance of I/O Buffer</td>
<td>10</td>
<td>pF</td>
<td>fC = 1MHz, TA = 25°C</td>
<td></td>
</tr>
</tbody>
</table>

See page 6 for Notes.
Note 1: Vol is degraded when the 8032AH/8052AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8032AH/8052AH as possible.

<table>
<thead>
<tr>
<th>Datum</th>
<th>Emitting Ports</th>
<th>Degraded I/O Lines</th>
<th>VOL (peak) (max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>A8–15, AD0–7</td>
<td>P1, Control Functions</td>
<td>0.8V</td>
</tr>
<tr>
<td>Write Data</td>
<td>AD0–7</td>
<td>P1, Control Functions, ALE</td>
<td>0.8V</td>
</tr>
</tbody>
</table>

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Variable Clock $f = 3.5$ MHz to 12 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>TCLCL</td>
<td>Oscillator Period</td>
<td>83.3</td>
</tr>
<tr>
<td>TCHCX</td>
<td>High Time</td>
<td>20</td>
</tr>
<tr>
<td>TCLCX</td>
<td>Low Time</td>
<td>20</td>
</tr>
<tr>
<td>TCLCH</td>
<td>Rise Time</td>
<td>20</td>
</tr>
<tr>
<td>TCHCL</td>
<td>Fall Time</td>
<td>20</td>
</tr>
</tbody>
</table>
AC CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$, $V_{SS} = 0V$, $C_L$ for $AD0-7$, ALE and $PSEN$ Outputs = $100$ pF, $C_L$ for all other outputs = $80$ pF

### PROGRAM MEMORY CHARACTERISTICS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Clock</th>
<th>Variable Clock $1/TCLCL = 3.5$ MHz to $12$ MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Min</strong></td>
<td><strong>Max</strong></td>
<td><strong>Unit</strong></td>
</tr>
<tr>
<td>TLHLL</td>
<td>ALE Pulse Width</td>
<td>127</td>
<td>ns</td>
</tr>
<tr>
<td>TAVLL</td>
<td>Address Setup to ALE</td>
<td>43</td>
<td>ns</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold After ALE</td>
<td>48</td>
<td>ns</td>
</tr>
<tr>
<td>TLLIV</td>
<td>ALE to Valid Instr In</td>
<td>233</td>
<td>ns</td>
</tr>
<tr>
<td>TLLPL</td>
<td>ALE To PSEN</td>
<td>58</td>
<td>ns</td>
</tr>
<tr>
<td>TPLPH</td>
<td>PSEN Pulse Width</td>
<td>215</td>
<td>ns</td>
</tr>
<tr>
<td>TPLIV</td>
<td>PSEN To Valid Instr In</td>
<td>125</td>
<td>ns</td>
</tr>
<tr>
<td>TPXIX</td>
<td>Input Instr Hold After PSEN</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TPXIZ</td>
<td>Input Instr Float After PSEN</td>
<td>63</td>
<td>ns</td>
</tr>
<tr>
<td>TPXAV</td>
<td>Address Valid After PSEN</td>
<td>75</td>
<td>ns</td>
</tr>
<tr>
<td>TAVIV</td>
<td>Address To Valid Instr In</td>
<td>302</td>
<td>ns</td>
</tr>
<tr>
<td>TAZPL</td>
<td>Address Float To PSEN</td>
<td>0</td>
<td>ns</td>
</tr>
</tbody>
</table>

### EXTERNAL DATA MEMORY CHARACTERISTICS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Clock</th>
<th>Variable Clock $1/TCLCL = 3.5$ MHz to $12$ MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Min</strong></td>
<td><strong>Max</strong></td>
<td><strong>Unit</strong></td>
</tr>
<tr>
<td>TRLRH</td>
<td>RD Pulse Width</td>
<td>400</td>
<td>ns</td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR Pulse Width</td>
<td>400</td>
<td>ns</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold After ALE</td>
<td>48</td>
<td>ns</td>
</tr>
<tr>
<td>TRLDV</td>
<td>RD To Valid Data In</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>TRHDX</td>
<td>Data Hold After RD</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TRHZD</td>
<td>Data Float After RD</td>
<td>97</td>
<td>ns</td>
</tr>
<tr>
<td>TLLDV</td>
<td>ALE To Valid Data In</td>
<td>517</td>
<td>ns</td>
</tr>
<tr>
<td>TAVDV</td>
<td>Address To Valid Data In</td>
<td>585</td>
<td>ns</td>
</tr>
<tr>
<td>TLLWL</td>
<td>ALE To WR or RD</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>TAVWL</td>
<td>Address To SR or RD</td>
<td>203</td>
<td>ns</td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR or RD High To ALE High</td>
<td>43</td>
<td>123</td>
</tr>
<tr>
<td>TDVWX</td>
<td>Data Valid To WR Transition</td>
<td>23</td>
<td>ns</td>
</tr>
<tr>
<td>TQVWH</td>
<td>Data Setup Before WR</td>
<td>433</td>
<td>ns</td>
</tr>
<tr>
<td>TWHQX</td>
<td>Data Hold After WR</td>
<td>33</td>
<td>ns</td>
</tr>
<tr>
<td>TRLAZ</td>
<td>Address Float After RD</td>
<td>0</td>
<td>ns</td>
</tr>
</tbody>
</table>
AC TIMING DIAGRAMS

EXTERNAL PROGRAM MEMORY READ CYCLE

- ALE
- PSEN
- A0–7 INSTR IN
- A8–15 ADDRESS OR SFR-P2

EXTERNAL DATA MEMORY READ CYCLE

- ALE
- PSEN
- RD
- A0–7
- A8–15 ADDRESS OR SFR-P2

EXTERNAL DATA MEMORY WRITE CYCLE

- ALE
- PSEN
- WR
- A0–7
- A8–15 ADDRESS OR SFR-P2

AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS

AC inputs during testing are driven at 2.4V for a logic “1” and 0.45V for a logic “0”. Timing measurements are made at 2.0V for a logic “1” and 0.8V for a logic “0”. For timing purposes, the float state is defined as the point at which an A0–7 pin sinks 2.4mA or sources 400μA at the voltage test levels.
CLOCK WAVEFORMS

<table>
<thead>
<tr>
<th>INTERNAL</th>
<th>STATE 4</th>
<th>STATE 5</th>
<th>STATE 6</th>
<th>STATE 1</th>
<th>STATE 2</th>
<th>STATE 3</th>
<th>STATE 4</th>
<th>STATE 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL 2</td>
<td>P1</td>
<td>P2</td>
<td>P1</td>
<td>P2</td>
<td>P1</td>
<td>P2</td>
<td>P1</td>
<td>P2</td>
</tr>
<tr>
<td>ALE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EXTERNAL PROGRAM MEMORY FETCH

PSEN

AD0-7

AD0-7

READ CYCLE

RD

00H IS EMITTED DURING THIS PERIOD

PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)

WR

PCL OUT (EVEN IF PROGRAM MEMORY IS INTERNAL)

WRITE CYCLE

A8-15

A8-15

PORT OPERATION

MOV PORT, SRC

MOV DEST, P1

(INCLUDES INTO, INT1, T0, T1)

SERIAL PORT SHIFT CLOCK

TXD (MODE 0)

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (TA = 25°C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.
80C51BH/80C51BH-2
CHMOS SINGLE COMPONENT 8-BIT MICROCONTROLLER
with Factory Mask-Programmable ROM

80C31BH/80C31BH-2
CHMOS SINGLE COMPONENT 8-BIT CONTROL-ORIENTED
CPU WITH RAM AND I/O

80C51BH/80C31BH — 3.5 to 12 MHz \( V_{CC} = 5V \pm 20\% \)
80C51BH-2/80C31BH-2 — 0.5 to 12 MHz \( V_{CC} = 5V \pm 20\% \)

- 128 X 8 RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- 64K Program Memory Space
- Boolean Processor
- 5 Interrupt Sources
- Programmable Serial Port
- 64K Data Memory Space

The MCS®-51 CHMOS products are fabricated on Intel’s advanced CHMOS III process and are functionally compatible with the standard MCS-51 HMOS and EPROM products. CHMOS III is a technology which combines the high speed and density characteristics of HMOS with the low power attributes of CMOS. This combination expands the effectiveness of the powerful MCS-51 architecture and instruction set.

Like the MCS-51 EPROM and HMOS, the MCS-51 CHMOS products have the following features: 4K of ROM (80C51BH/80C51BH-2 only); 128 bytes of RAM; 32 I/O lines; two 16-bit timer/counters; a five-source two-level interrupt structure; a full duplex serial port; and on-chip oscillator and clock circuitry. In addition, the MCS-51 CHMOS products have two software selectable modes of reduced activity for further power reduction — Idle and Power Down.

Idle mode freezes the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. Power Down mode saves the RAM contents but freezes the oscillator causing all other chip functions to be inoperative.

Figure 1. Block Diagram
IDLE AND POWER DOWN OPERATION

Figure 3 shows the internal Idle and Power Down clock configuration. As illustrated, Power Down operation freezes the oscillator. Idle mode operation allows the interrupt, serial port, and timer blocks to continue to function while the clock to the CPU is halted.

These special modes are activated by software via the Special Function Register, PCON. Its hardware address is 87H. PCON is not bit addressable.

**PCON: Power Control Register**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMOD</td>
<td>PCON.7</td>
<td>Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.</td>
</tr>
<tr>
<td>—</td>
<td>PCON.6</td>
<td>(Reserved)</td>
</tr>
<tr>
<td>—</td>
<td>PCON.5</td>
<td>(Reserved)</td>
</tr>
<tr>
<td>—</td>
<td>PCON.4</td>
<td>(Reserved)</td>
</tr>
<tr>
<td>GF1</td>
<td>PCON.3</td>
<td>General-purpose flag bit.</td>
</tr>
<tr>
<td>GF0</td>
<td>PCON.2</td>
<td>General-purpose flag bit.</td>
</tr>
<tr>
<td>PD</td>
<td>PCON.1</td>
<td>Power Down bit. Setting this bit activates power down operation.</td>
</tr>
<tr>
<td>IDL</td>
<td>PCON.0</td>
<td>Idle mode bit. Setting this bit activates idle mode operation.</td>
</tr>
</tbody>
</table>

If 1's are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).
Idle Mode

The instruction that sets PCON.0 is the last instruction executed in the normal operating mode before Idle mode is activated. Once in the Idle mode, the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all other registers maintain their data during Idle. Table 1 describes the status of the external pins during Idle mode.

There are two ways to terminate the Idle mode. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating Idle mode. The interrupt is serviced, and following RETI, the next instruction to be executed will be the one following the instruction that wrote a 1 to PCON.0.

The flag bits GFO and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an enabled interrupt, the service routine can examine the status of the flag bits.

The second way of terminating the Idle mode is with a hardware reset. Since the oscillator is still running, the hardware reset needs to be active for only 2 machine cycles (24 oscillator periods) to complete the reset operation.

Power Down Mode

The instruction that sets PCON.1 is the last executed prior to going into power down. Once in power down, the oscillator is stopped. Only the contents of the on-chip RAM is preserved. The Special Function Registers are not saved. A hardware reset is the only way of exiting the power down mode.

In the Power Down mode, VCC may be lowered to minimize circuit power consumption. Care must be taken to ensure the voltage is not reduced until the power down mode is entered, and that the voltage is restored before the hardware reset is applied which frees the oscillator. Reset should not be released until the oscillator has restarted and stabilized.

Table 1 describes the status of the external pins while in the power down mode. It should be noted that if the power down mode is activated while in external program memory, the port data that is held in the Special Function Register P2 is restored to Port 2. If the data is a 1, the port pin is held high during the power down mode by the strong pullup, p1, shown in figure 4.

Table 1. Status of the external pins during Idle and Power Down modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Program Memory</th>
<th>ALE</th>
<th>PSEN</th>
<th>PORT0</th>
<th>PORT1</th>
<th>PORT2</th>
<th>PORT3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Internal</td>
<td>1</td>
<td>1</td>
<td>Port Data</td>
<td>Port Data</td>
<td>Port Data</td>
<td>Port Data</td>
</tr>
<tr>
<td>Idle</td>
<td>Internal</td>
<td>0</td>
<td>0</td>
<td>Port Data</td>
<td>Port Data</td>
<td>Port Data</td>
<td>Port Data</td>
</tr>
<tr>
<td>Power Down</td>
<td>Internal</td>
<td>0</td>
<td>0</td>
<td>Floating</td>
<td>Port Data</td>
<td>Address</td>
<td>Port Data</td>
</tr>
<tr>
<td>Power Down</td>
<td>External</td>
<td>0</td>
<td>0</td>
<td>Floating</td>
<td>Port Data</td>
<td>Port Data</td>
<td>Port Data</td>
</tr>
</tbody>
</table>

80C51BH I/O Ports

The I/O port drive of the 80C51BH is similar to the 8051. The I/O buffers for Ports 1, 2, and 3 are implemented as shown in figure 4.

When the port latch contains a 0, all pFETS in figure 4 are off while the nFET is turned on. When the port latch makes a 0-to-1 transition, the nFET turns off. The strong pullup pFET, p1, turns on for two oscillator periods, pulling the output high very rapidly. As the output line is drawn high, pFET p3 turns on through the inverter to supply the IOH source current. This inverter and p3 form a latch which holds the 1 and is supported by p2.

When Port 2 is used as an address port, for access to external program or data memory, any address bit that contains a 1 will have its strong pullup turned on for the entire duration of the external memory access.

When an I/O pin on Ports 1, 2, or 3 is used as an input, the user should be aware that the external circuit must sink current during the logical 1-to-0 transition. The maximum sink current is specified as ITL under the D.C. Specifications. When the input goes below approximately 2V, p3 turns off to save ICC current. Note, when returning to a logical 1, p2 is the only internal pullup that is on. This will result in a slow rise time if the user’s circuit does not force the input.
Figure 4. I/O Buffers in the 80C51BH (Ports 1, 2, 3)

PIN DESCRIPTIONS

VCC
Supply voltage during normal, Idle, and Power Down operations.

VSS
Circuit ground.

Port 0
Port 0 is an 8-bit open drain bi-directional I/O port. Port 0 pins that have 1's written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1's. Port 0 also outputs the code bytes during program verification in the 80C51BH. External pullups are required during program verification.

Port 1
Port 1 is an 8-bit bi-directional I/O port with internal pullups. Port 1 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during program verification.

Port 2
Port 2 is an 8-bit bi-directional I/O port with internal pullups. Port 2 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @DPTR). In this application it uses strong internal pullups when emitting 1's. During accesses to external Data Memory that use 8-bit addresses (MOVX @Ri), Port 2 emits the contents of the P2 Special Function Register.

Port 3
Port 3 is an 8-bit bi-directional I/O port with internal pullups. Port 3 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups.
Port 3 also serves the functions of various special features of the MCS-51 Family, as listed below:

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RXD (serial input port)</td>
</tr>
<tr>
<td>P3.1</td>
<td>TXD (serial output port)</td>
</tr>
<tr>
<td>P3.2</td>
<td>INTO (external interrupt 0)</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 (external interrupt 1)</td>
</tr>
<tr>
<td>P3.4</td>
<td>TO (Timer 0 external input)</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 (Timer 1 external input)</td>
</tr>
<tr>
<td>P3.6</td>
<td>WR (external data memory write strobe)</td>
</tr>
<tr>
<td>P3.7</td>
<td>RD (external data memory read strobe)</td>
</tr>
</tbody>
</table>

**RST**

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. An internal diffused resistor to VSS permits Power-On reset using only an external capacitor to VCC.

**ALE**

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory.

In normal operation ALE is emitted at a constant rate of \( \frac{1}{4} \) the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

**PSEN**

Program Store Enable is the read strobe to external Program Memory.

When the 80C51BH is executing code from external Program Memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external Data Memory.

**EA**

External Access enable. EA must be externally held low in order to enable the device to fetch code from external Program Memory locations 0000H to 0FFFH.

**XTAL1**

Input to the inverting oscillator amplifier and input to the internal clock generator circuits.

**XTAL2**

Output from the inverting oscillator amplifier.

**OSCILLATOR CHARACTERISTICS**

XTAL1 and XTAL2 are the input and output respectively, of an inverting amplifier which is configured for use as an on-chip oscillator, as shown in Figure 5. Either a quartz crystal or ceramic resonator may be used. More detailed information concerning the use of the on-chip oscillator is available in Application Note AP-155, “Oscillators for Microcontrollers.”

To drive the device from an external clock source, XTAL1 should be driven while XTAL2 is left unconnected as shown in figure 6. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum high and low times specified on the Data Sheet must be observed.

![Figure 5. Crystal Oscillator](image-url)

![Figure 6. External Drive Configuration](image-url)
# Absolute Maximum Ratings

<table>
<thead>
<tr>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambient Temperature Under Bias</td>
<td>0°C</td>
<td>70°C</td>
<td>°C</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>-65°C</td>
<td>+150°C</td>
<td>°C</td>
</tr>
<tr>
<td>Voltage on Any Pin to VSS</td>
<td>-0.5V</td>
<td>VCC + 1V</td>
<td>V</td>
</tr>
<tr>
<td>Voltage on VCC to VSS</td>
<td>-0.5V</td>
<td>VCC + 7V</td>
<td>V</td>
</tr>
<tr>
<td>Power Dissipation</td>
<td></td>
<td>1.0W</td>
<td></td>
</tr>
</tbody>
</table>

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## D.C. Characteristics

\( \text{TA} = 0°C \text{ to } 70°C; \ VSS = 0V; \ VCC = 5V \pm 20\% \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage (Except XTAL1, RST)</td>
<td>-0.5</td>
<td>0.2VCC -.1</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>0.2VCC+.9</td>
<td>VCC+.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage to XTAL1, RST</td>
<td>0.7VCC</td>
<td>VCC+.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage (Ports 1, 2, 3)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 1.6 mA</td>
<td></td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage (Port 0, ALE, PSEN)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 3.2 mA (Note 1)</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage (Ports 1, 2, 3)</td>
<td>2.4</td>
<td>V</td>
<td>IOH = -80µA VCC = 5V ± 10%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.75VCC</td>
<td>V</td>
<td>IOH = -30µA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.9VCC</td>
<td>V</td>
<td>IOH = -10µA</td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage (Port 0 in External Bus Mode, ALE, PSEN)</td>
<td>2.4</td>
<td>V</td>
<td>IOH = -400µA VCC = 5V ± 10%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.75VCC</td>
<td>V</td>
<td>IOH = -150µA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.9VCC</td>
<td>V</td>
<td>IOH = -40µA (Note 2)</td>
</tr>
<tr>
<td>IIL</td>
<td>Logical 0 Input Current (Ports 1, 2, 3)</td>
<td>-50</td>
<td>µA</td>
<td>Vin = 0.45V</td>
<td></td>
</tr>
<tr>
<td>ITL</td>
<td>Logical 1 to 0 transition Current (Ports 1, 2, 3)</td>
<td>-500</td>
<td>µA</td>
<td>Vin = 2.0V</td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Input Leakage Current (Port 0, EA)</td>
<td>±10</td>
<td>µA</td>
<td>0.45 &lt; Vin &lt; VCC</td>
<td></td>
</tr>
<tr>
<td>RRST</td>
<td>RST Pulldown Resistor</td>
<td>40</td>
<td>125</td>
<td>Kohm</td>
<td></td>
</tr>
<tr>
<td>CIO</td>
<td>Pin Capacitance</td>
<td>10</td>
<td>pF</td>
<td>test freq = 1 MHz, TA = 25°C</td>
<td></td>
</tr>
<tr>
<td>IPD</td>
<td>Power Down Current</td>
<td>50</td>
<td>µA</td>
<td>VCC = 2 to 6V (Note 3)</td>
<td></td>
</tr>
</tbody>
</table>

### Maximum Operating \( I_{CC} \) (mA) (note 4)

<table>
<thead>
<tr>
<th>VCC</th>
<th>4V</th>
<th>5V</th>
<th>6V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5 MHz</td>
<td>1.6</td>
<td>2.2</td>
<td>3</td>
</tr>
<tr>
<td>3.5 MHz</td>
<td>4.3</td>
<td>5.7</td>
<td>7.5</td>
</tr>
<tr>
<td>8 MHz</td>
<td>8.3</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>12 MHz</td>
<td>12</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>

### Maximum Idle \( I_{CC} \) (mA) (note 5)

<table>
<thead>
<tr>
<th>VCC</th>
<th>4V</th>
<th>5V</th>
<th>6V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5 MHz</td>
<td>0.6</td>
<td>0.9</td>
<td>1.2</td>
</tr>
<tr>
<td>3.5 MHz</td>
<td>1.1</td>
<td>1.6</td>
<td>2.2</td>
</tr>
<tr>
<td>8 MHz</td>
<td>1.8</td>
<td>2.7</td>
<td>3.7</td>
</tr>
<tr>
<td>12 MHz</td>
<td>2.5</td>
<td>3.7</td>
<td>5</td>
</tr>
</tbody>
</table>
Note 1: Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the VOLS of ALE, and Ports 1 and 3. The noise is due to the external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make a 1-to-0 transition during bus operations. In the worst case (capacitive loading > 100 pF), the noise pulse on ALE line may exceed 0.8V. In such cases it may be desirable to qualify ALE with a Schmitt Trigger, or use an address latch with a Schmitt Trigger STROBE input.

Note 2: Capacitive loading on Ports 0 and 2 may cause the VOH on ALE and PSEN to momentarily fall below the .9VCC specification when the address bits are stabilizing.

Note 3: Power Down ICC is measured with all output pins disconnected; EA = PORT0 = VCC; XTAL2 N.C.; RST = VSS.

Note 4: ICC is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 10 ns, Vll = VSS+.5v, Vih = VCC-.5v; XTAL2 N.C.; EA = RST = PORT0 = VCC.

Note 5: Idle ICC is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 10 ns, Vll = VSS+.5v, Vih = VCC-.5v; XTAL2 N.C.; EA = PORT0 = VCC; RST = VSS.

A.C. CHARACTERISTICS (TA = 0°C to 70°C; VSS = 0V; VCC = 5V ± 20%; Load Capacitance for Port 0, ALE, and PSEN = 100 pF, Load Capacitance for All Other Outputs = 80 pF)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/TCLCL</td>
<td>Oscillator Freq (80C51BH)</td>
<td>3.5</td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>Oscillator Freq (80C51BH-2)</td>
<td>0.5</td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td>TLHLL</td>
<td>ALE Pulse Width</td>
<td>2TCLCL-40</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TAVLL</td>
<td>Address Valid to ALE Low</td>
<td>TCLCL-40</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold After ALE Low</td>
<td>TCLCL-35</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TLLIV</td>
<td>ALE Low to Valid Instr In</td>
<td>4TCLCL-150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TLLPL</td>
<td>ALE Low to PSEN Low</td>
<td>TCLCL-25</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TPLPH</td>
<td>PSEN Pulse Width</td>
<td>3TCLCL-35</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TPLIV</td>
<td>PSEN Low to Valid Instr In</td>
<td>3TCLCL-150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TPXIX</td>
<td>Input Instr Hold After PSEN</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TPXIZ</td>
<td>Input Instr Float After PSEN</td>
<td>TCLCL-20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TPXAV</td>
<td>PSEN to Address Valid</td>
<td>TCLCL-8</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TAVIV</td>
<td>Address to Valid Instr In</td>
<td>5TCLCL-150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TPLAZ</td>
<td>PSEN Low to Address Float</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TRLRH</td>
<td>RD Pulse Width</td>
<td>6TCLCL-100</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR Pulse Width</td>
<td>6TCLCL-100</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TRLDV</td>
<td>RD Low to Valid Data In</td>
<td>5TCLCL-165</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TRHDX</td>
<td>Data Hold After RD</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TRHDZ</td>
<td>Data Float After RD</td>
<td>2TCLCL-70</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TLLDV</td>
<td>ALE Low to Valid Data In</td>
<td>8TCLCL-150</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>
EXTERNAL PROGRAM MEMORY READ CYCLE

- ALE
- PSEN
- PORT 0
- PORT 2

EXTERNAL DATA MEMORY READ CYCLE

- ALE
- PSEN
- RD
- PORT 0
- PORT 2

9-31
### A.C. CHARACTERISTICS

#### EXTERNAL PROGRAM AND DATA MEMORY CHARACTERISTICS (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAVDV</td>
<td>Address to Valid Data In</td>
<td>9TCLCL - 165</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TLLWL</td>
<td>ALE Low to RD or WR Low</td>
<td>3TCLCL - 50</td>
<td>3TCLCL + 50</td>
<td>ns</td>
</tr>
<tr>
<td>TVWL</td>
<td>Address to RD or WR Low</td>
<td>4TCLCL - 130</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TQVWX</td>
<td>Data Valid to WR Transition</td>
<td>TCLCL - 60</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TQVWH</td>
<td>Data Valid to WR High</td>
<td>7TCLCL - 150</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TVHQX</td>
<td>Data Held After WR</td>
<td>TCLCL - 50</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TRLAZ</td>
<td>RD Low to Address Float</td>
<td>TCLCL - 40</td>
<td>TCLCL + 50</td>
<td>ns</td>
</tr>
<tr>
<td>TWHHLH</td>
<td>RD or WR High to ALE High</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### EXTERNAL CLOCK DRIVE

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/TCLCL</td>
<td>Oscillator Freq (80C51BH)</td>
<td>3.5</td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td></td>
<td>Oscillator Freq (80C51BH-2)</td>
<td>0.5</td>
<td>12</td>
<td>MHz</td>
</tr>
<tr>
<td>TCHCX</td>
<td>High Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCLCX</td>
<td>Low Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCLCH</td>
<td>Rise Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCHCL</td>
<td>Fall Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

---

![Diagram](image-url)
EXTERNAL DATA MEMORY WRITE CYCLE

SERIAL PORT TIMING — SHIFT REGISTER MODE

A.C. CHARACTERISTICS: \( T_A = 0^\circ C \) to 70°C; \( V_{SS} = 0V; V_{CC} = 5V \pm 20%; \) Load Capacitance = 80 pF

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXLXL</td>
<td>Serial Port Clock Cycle Time</td>
<td>12TCLCL</td>
<td></td>
<td>( \mu S )</td>
</tr>
<tr>
<td>TQVXH</td>
<td>Output Data Setup to Clock Rising Edge</td>
<td>10TCLCL - 133</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TXHQX</td>
<td>Output Data Hold After Clock Rising Edge</td>
<td>2TCLCL - 117</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TXHDX</td>
<td>Input Data Hold After Clock Rising Edge</td>
<td>0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TXHDV</td>
<td>Clock Rising Edge to Input Data Valid</td>
<td>10TCLCL - 133</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

SHIFT REGISTER TIMING WAVEFORMS
### Table 2. MCS®-51 Instruction Set Description

#### ARITHMETIC OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A,Rn</td>
<td>Add register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A,direct</td>
<td>Add direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ADD A,@Ri</td>
<td>Add indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A,#data</td>
<td>Add immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,Rn</td>
<td>Add register to Accumulator with Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,direct</td>
<td>Add direct byte to A with Carry flag</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,@Ri</td>
<td>Add indirect RAM to A with Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,#data</td>
<td>Add immediate data to A with Carry flag</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SUBB A,Rn</td>
<td>Subtract register from A with Borrow</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SUBB A,direct</td>
<td>Subtract direct byte from A with Borrow</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SUBB A,@Ri</td>
<td>Subtract indirect RAM from A with Borrow</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SUBB A,#data</td>
<td>Subtract immediate data from A with Borrow</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC Rn</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC direct</td>
<td>Increment direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>INC @Ri</td>
<td>Increment indirect RAM</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC DPTR</td>
<td>Increment Data Pointer</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC Rn</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC direct</td>
<td>Decrement direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>DEC @Ri</td>
<td>Decrement indirect RAM</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MUL AB</td>
<td>Multiply A &amp; B</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>DIV AB</td>
<td>Divide A by B</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal Adjust Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### LOGICAL OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Destination</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL A,Rn</td>
<td>AND register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A,direct</td>
<td>AND direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ANL A,@Ri</td>
<td>AND indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A,#data</td>
<td>AND immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ANL direct,A</td>
<td>AND Accumulator to direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ANL direct,#data</td>
<td>AND immediate data to direct byte</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ORL A,Rn</td>
<td>OR register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A,direct</td>
<td>OR direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### LOGICAL OPERATIONS (CONTINUED)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Destination</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORL A,@Ri</td>
<td>OR indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A,#data</td>
<td>OR immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ORL direct,A</td>
<td>OR Accumulator to direct byte</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL direct,#data</td>
<td>OR immediate data to direct byte</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>XRL A,Rn</td>
<td>Exclusive-OR register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A,direct</td>
<td>Exclusive-OR direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>XRL A,@Ri</td>
<td>Exclusive-OR indirect RAM to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A,#data</td>
<td>Exclusive-OR immediate data to A</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>XRL direct,A</td>
<td>Exclusive-OR Accumulator to direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>XRL direct,#data</td>
<td>Exclusive-OR immediate data to direct byte</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate Accumulator Left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A Left through Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate Accumulator Right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A Right through Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles within the Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### DATA TRANSFER

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Rn</td>
<td>Move register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,direct</td>
<td>Move direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,@Ri</td>
<td>Move indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A,#data</td>
<td>Move immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV Rn,A</td>
<td>Move Accumulator to register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV Rn,direct</td>
<td>Move direct byte to register</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV Rn,#data</td>
<td>Move immediate data to register</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV direct,A</td>
<td>Move Accumulator to direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV direct,Rn</td>
<td>Move register to direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV direct,direct</td>
<td>Move direct byte to direct</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV direct,@Ri</td>
<td>Move indirect RAM to direct byte</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 2. MCS-51 Instruction Set Description (Continued)

<table>
<thead>
<tr>
<th>DATA TRANSFER (CONTINUED)</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>direct,#data</td>
<td>Move immediate data to direct byte</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>MOV</td>
<td>@Ri,A</td>
<td>Move Accumulator to indirect RAM</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>@Ri,direct</td>
<td>Move direct byte to indirect RAM</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV</td>
<td>@Ri,#data</td>
<td>Move immediate data to indirect RAM</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV</td>
<td>DPTR,#data16</td>
<td>Load Data Pointer with a 16-bit constant</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>MOV C @A+DPTR</td>
<td>Move Code byte relative to DPTR to A</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MOV C @A+PC</td>
<td>Move Code byte relative to PC to A</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MOVX A,@Ri</td>
<td>Move External RAM (8-bit addr) to A</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MOVX A,DPTR</td>
<td>Move External RAM (16-bit addr) to A</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MOVX @Ri,A</td>
<td>Move A to External RAM (8-bit addr)</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MOVX DPTR,A</td>
<td>Move A to External RAM (16-bit addr)</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>PUSH direct</td>
<td>Push direct byte onto stack</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>POP direct</td>
<td>Pop direct byte from stack</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>XCH A,Rn</td>
<td>Exchange register with Accumulator</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>XCH A,direct</td>
<td>Exchange direct byte with Accumulator</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>XCH A,Ri</td>
<td>Exchange indirect RAM with A</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>XCHD A,Ri</td>
<td>Exchange low-order Digit ind RAM w A</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BOOLEAN VARIABLE MANIPULATION</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>Clear Carry flag</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CLR bit</td>
<td>Clear direct bit</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SETB C</td>
<td>Set Carry flag</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SETB bit</td>
<td>Set direct Bit</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement Carry flag</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CPL bit</td>
<td>Complement direct bit</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ANL C,bit</td>
<td>AND direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ANL C/bit</td>
<td>AND complement of direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ORL C/bit</td>
<td>OR direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ORL C/bit</td>
<td>OR complement of direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MOV C/bit</td>
<td>Move direct bit to Carry flag</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>MOV bit,c</td>
<td>Move Carry flag to direct bit</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROGRAM AND MACHINE CONTROL</th>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACALL addr11</td>
<td>Absolute Subroutine Call</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>LCALL addr16</td>
<td>Long Subroutine Call</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>Return from subroutine</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>RETI</td>
<td>Return from interrupt</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>AJMP addr11</td>
<td>Absolute Jump</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>LJMP addr16</td>
<td>Long Jump</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SJMP rel</td>
<td>Short Jump (relative addr)</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JMP @A+DPTR</td>
<td>Jump indirect relative to the DPTR</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JZ rel</td>
<td>Jump if Accumulator is Zero</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JNZ rel</td>
<td>Jump if Accumulator is Not Zero</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JC rel</td>
<td>Jump if Carry flag is set</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JNC rel</td>
<td>Jump if No Carry flag</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JB bit,rel</td>
<td>Jump if direct Bit set</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JNB bit,rel</td>
<td>Jump if direct Bit Not set</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JBC bit,rel</td>
<td>Jump if direct Bit is set &amp; Clear bit</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CJNE A,direct,rel</td>
<td>Compare direct to A &amp; Jump if Not Equal</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CJNE A,#data,rel</td>
<td>Comp, immed, to A &amp; Jump if Not Equal</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>CJNE Rn,#data,rel</td>
<td>Comp, immed, to reg &amp; Jump if Not Equal</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DJNZ Rn,rel</td>
<td>Decrement register &amp; Jump if Not Zero</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DJNZ direct,rel</td>
<td>Decrement direct &amp; Jump if Not Zero</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Notes on data addressing modes:
- Rn — Working register R0-R7
- direct — 128 internal RAM locations, any I/O port, control or status register
- @Ri — Indirect internal RAM location addressed by register R0 or R1
- #data — 8-bit constant included in instruction
- #data16 — 16-bit constant included as bytes 2 & 3 of instruction
- bit — 128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:
- addr16 — Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
- Addr11 — Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
- rel — SJMP and all conditional jumps include an 8-bit offset byte, Range is +127-128 bytes relative to first byte of the following instruction

All mnemonics copyrighted © Intel Corporation 1979
Table 3. Instruction Opcodes in Hexadecimal Order

<table>
<thead>
<tr>
<th>Hex Code</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>02</td>
<td>3</td>
<td>LJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>03</td>
<td>1</td>
<td>RR</td>
<td>A</td>
</tr>
<tr>
<td>04</td>
<td>1</td>
<td>INC</td>
<td>A</td>
</tr>
<tr>
<td>05</td>
<td>2</td>
<td>INC</td>
<td>data addr</td>
</tr>
<tr>
<td>06</td>
<td>1</td>
<td>INC</td>
<td>@R0</td>
</tr>
<tr>
<td>07</td>
<td>1</td>
<td>INC</td>
<td>@R1</td>
</tr>
<tr>
<td>08</td>
<td>1</td>
<td>INC</td>
<td>R0</td>
</tr>
<tr>
<td>09</td>
<td>1</td>
<td>INC</td>
<td>R1</td>
</tr>
<tr>
<td>0A</td>
<td>1</td>
<td>INC</td>
<td>R2</td>
</tr>
<tr>
<td>0B</td>
<td>1</td>
<td>INC</td>
<td>R3</td>
</tr>
<tr>
<td>0C</td>
<td>1</td>
<td>INC</td>
<td>R4</td>
</tr>
<tr>
<td>0D</td>
<td>1</td>
<td>INC</td>
<td>R5</td>
</tr>
<tr>
<td>0E</td>
<td>1</td>
<td>INC</td>
<td>R6</td>
</tr>
<tr>
<td>0F</td>
<td>1</td>
<td>INC</td>
<td>R7</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>JBC</td>
<td>bit addr, code addr</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>LCALL</td>
<td>code addr</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>RRC</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>DEC</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>DEC</td>
<td>data addr</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>DEC</td>
<td>@R0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>DEC</td>
<td>@R1</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>DEC</td>
<td>R0</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>DEC</td>
<td>R1</td>
</tr>
<tr>
<td>1A</td>
<td>1</td>
<td>DEC</td>
<td>R2</td>
</tr>
<tr>
<td>1B</td>
<td>1</td>
<td>DEC</td>
<td>R3</td>
</tr>
<tr>
<td>1C</td>
<td>1</td>
<td>DEC</td>
<td>R4</td>
</tr>
<tr>
<td>1D</td>
<td>1</td>
<td>DEC</td>
<td>R5</td>
</tr>
<tr>
<td>1E</td>
<td>1</td>
<td>DEC</td>
<td>R6</td>
</tr>
<tr>
<td>1F</td>
<td>1</td>
<td>DEC</td>
<td>R7</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>JB</td>
<td>bit addr, code addr</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>RL</td>
<td>A</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>ADD</td>
<td>A,#data</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>ADD</td>
<td>A.data addr</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>ADD</td>
<td>A,@R0</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>ADD</td>
<td>A,@R1</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>ADD</td>
<td>A,R0</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>ADD</td>
<td>A,R1</td>
</tr>
<tr>
<td>2A</td>
<td>1</td>
<td>ADD</td>
<td>A,R2</td>
</tr>
<tr>
<td>2B</td>
<td>1</td>
<td>ADD</td>
<td>A,R3</td>
</tr>
<tr>
<td>2C</td>
<td>1</td>
<td>ADD</td>
<td>A,R4</td>
</tr>
<tr>
<td>2D</td>
<td>1</td>
<td>ADD</td>
<td>A,R5</td>
</tr>
<tr>
<td>2E</td>
<td>1</td>
<td>ADD</td>
<td>A,R6</td>
</tr>
<tr>
<td>2F</td>
<td>1</td>
<td>ADD</td>
<td>A,R7</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>JNB</td>
<td>bit addr, code addr</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>RETI</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>1</td>
<td>RLC</td>
<td>A</td>
</tr>
<tr>
<td>34</td>
<td>2</td>
<td>ADDC</td>
<td>A,#data</td>
</tr>
<tr>
<td>35</td>
<td>2</td>
<td>ADDC</td>
<td>A.data addr</td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>ADDC</td>
<td>A,@R0</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>ADDC</td>
<td>A,@R1</td>
</tr>
<tr>
<td>38</td>
<td>1</td>
<td>ADDC</td>
<td>A,R0</td>
</tr>
<tr>
<td>39</td>
<td>1</td>
<td>ADDC</td>
<td>A,R1</td>
</tr>
<tr>
<td>3A</td>
<td>1</td>
<td>ADDC</td>
<td>A,R2</td>
</tr>
<tr>
<td>3B</td>
<td>1</td>
<td>ADDC</td>
<td>A,R3</td>
</tr>
<tr>
<td>3C</td>
<td>1</td>
<td>ADDC</td>
<td>A,R4</td>
</tr>
<tr>
<td>3D</td>
<td>1</td>
<td>ADDC</td>
<td>A,R5</td>
</tr>
<tr>
<td>3E</td>
<td>1</td>
<td>ADDC</td>
<td>A,R6</td>
</tr>
<tr>
<td>3F</td>
<td>1</td>
<td>ADDC</td>
<td>A,R7</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>JC</td>
<td>code addr</td>
</tr>
<tr>
<td>41</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
<td>ORL</td>
<td>data addr,A</td>
</tr>
<tr>
<td>43</td>
<td>3</td>
<td>ORL</td>
<td>data addr,#data</td>
</tr>
<tr>
<td>44</td>
<td>2</td>
<td>ORL</td>
<td>A,#data</td>
</tr>
<tr>
<td>45</td>
<td>2</td>
<td>ORL</td>
<td>A.data addr</td>
</tr>
<tr>
<td>46</td>
<td>1</td>
<td>ORL</td>
<td>A,@R0</td>
</tr>
<tr>
<td>47</td>
<td>1</td>
<td>ORL</td>
<td>A,@R1</td>
</tr>
<tr>
<td>48</td>
<td>1</td>
<td>ORL</td>
<td>A,R0</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
<td>ORL</td>
<td>A,R1</td>
</tr>
<tr>
<td>4A</td>
<td>1</td>
<td>ORL</td>
<td>A,R2</td>
</tr>
<tr>
<td>4B</td>
<td>1</td>
<td>ORL</td>
<td>A,R3</td>
</tr>
<tr>
<td>4C</td>
<td>1</td>
<td>ORL</td>
<td>A,R4</td>
</tr>
<tr>
<td>4D</td>
<td>1</td>
<td>ORL</td>
<td>A,R5</td>
</tr>
<tr>
<td>4E</td>
<td>1</td>
<td>ORL</td>
<td>A,R6</td>
</tr>
<tr>
<td>4F</td>
<td>1</td>
<td>ORL</td>
<td>A,R7</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>JNC</td>
<td>code addr</td>
</tr>
<tr>
<td>51</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>52</td>
<td>2</td>
<td>ANL</td>
<td>data addr,A</td>
</tr>
<tr>
<td>53</td>
<td>3</td>
<td>ANL</td>
<td>data addr,#data</td>
</tr>
<tr>
<td>54</td>
<td>2</td>
<td>ANL</td>
<td>A,#data</td>
</tr>
<tr>
<td>55</td>
<td>2</td>
<td>ANL</td>
<td>A.data addr</td>
</tr>
<tr>
<td>56</td>
<td>1</td>
<td>ANL</td>
<td>A,@R0</td>
</tr>
<tr>
<td>57</td>
<td>1</td>
<td>ANL</td>
<td>A,@R1</td>
</tr>
<tr>
<td>58</td>
<td>1</td>
<td>ANL</td>
<td>A,R0</td>
</tr>
<tr>
<td>59</td>
<td>1</td>
<td>ANL</td>
<td>A,R1</td>
</tr>
<tr>
<td>5A</td>
<td>1</td>
<td>ANL</td>
<td>A,R2</td>
</tr>
<tr>
<td>5B</td>
<td>1</td>
<td>ANL</td>
<td>A,R3</td>
</tr>
<tr>
<td>5C</td>
<td>1</td>
<td>ANL</td>
<td>A,R4</td>
</tr>
<tr>
<td>5D</td>
<td>1</td>
<td>ANL</td>
<td>A,R5</td>
</tr>
<tr>
<td>5E</td>
<td>1</td>
<td>ANL</td>
<td>A,R6</td>
</tr>
<tr>
<td>5F</td>
<td>1</td>
<td>ANL</td>
<td>A,R7</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>JZ</td>
<td>code addr</td>
</tr>
<tr>
<td>61</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>62</td>
<td>2</td>
<td>XRL</td>
<td>data addr,A</td>
</tr>
<tr>
<td>63</td>
<td>3</td>
<td>XRL</td>
<td>data addr,#data</td>
</tr>
<tr>
<td>64</td>
<td>2</td>
<td>XRL</td>
<td>A,#data</td>
</tr>
<tr>
<td>65</td>
<td>2</td>
<td>XRL</td>
<td>A.data addr</td>
</tr>
</tbody>
</table>
### Table 3. Instruction Opcodes in Hexadecimal Order (Continued)

<table>
<thead>
<tr>
<th>Hex Code</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>1</td>
<td>XRL</td>
<td>A,@R0</td>
</tr>
<tr>
<td>67</td>
<td>1</td>
<td>XRL</td>
<td>A,@R1</td>
</tr>
<tr>
<td>68</td>
<td>1</td>
<td>XRL</td>
<td>A,R0</td>
</tr>
<tr>
<td>69</td>
<td>1</td>
<td>XRL</td>
<td>A,R1</td>
</tr>
<tr>
<td>6A</td>
<td>1</td>
<td>XRL</td>
<td>A,R2</td>
</tr>
<tr>
<td>6B</td>
<td>1</td>
<td>XRL</td>
<td>A,R3</td>
</tr>
<tr>
<td>6C</td>
<td>1</td>
<td>XRL</td>
<td>A,R4</td>
</tr>
<tr>
<td>6D</td>
<td>1</td>
<td>XRL</td>
<td>A,R5</td>
</tr>
<tr>
<td>6E</td>
<td>1</td>
<td>XRL</td>
<td>A,R6</td>
</tr>
<tr>
<td>6F</td>
<td>1</td>
<td>XRL</td>
<td>A,R7</td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>JNZ</td>
<td>code addr</td>
</tr>
<tr>
<td>71</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>72</td>
<td>2</td>
<td>ORL</td>
<td>C,bit addr</td>
</tr>
<tr>
<td>73</td>
<td>1</td>
<td>JMP</td>
<td>@A+DPTR</td>
</tr>
<tr>
<td>74</td>
<td>2</td>
<td>MOV</td>
<td>A, #data</td>
</tr>
<tr>
<td>75</td>
<td>3</td>
<td>MOV</td>
<td>data addr,#data</td>
</tr>
<tr>
<td>76</td>
<td>2</td>
<td>MOV</td>
<td>@R0,#data</td>
</tr>
<tr>
<td>77</td>
<td>2</td>
<td>MOV</td>
<td>@R1,#data</td>
</tr>
<tr>
<td>78</td>
<td>2</td>
<td>MOV</td>
<td>R0,#data</td>
</tr>
<tr>
<td>79</td>
<td>2</td>
<td>MOV</td>
<td>R1,#data</td>
</tr>
<tr>
<td>7A</td>
<td>2</td>
<td>MOV</td>
<td>R2,#data</td>
</tr>
<tr>
<td>7B</td>
<td>2</td>
<td>MOV</td>
<td>R3,#data</td>
</tr>
<tr>
<td>7C</td>
<td>2</td>
<td>MOV</td>
<td>R4,#data</td>
</tr>
<tr>
<td>7D</td>
<td>2</td>
<td>MOV</td>
<td>R5,#data</td>
</tr>
<tr>
<td>7E</td>
<td>2</td>
<td>MOV</td>
<td>R6,#data</td>
</tr>
<tr>
<td>7F</td>
<td>2</td>
<td>MOV</td>
<td>R7,#data</td>
</tr>
<tr>
<td>80</td>
<td>2</td>
<td>SJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>81</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>82</td>
<td>2</td>
<td>ANL</td>
<td>C, bit addr</td>
</tr>
<tr>
<td>83</td>
<td>1</td>
<td>MOVC</td>
<td>A,@A+PC</td>
</tr>
<tr>
<td>84</td>
<td>1</td>
<td>DIV</td>
<td>AB</td>
</tr>
<tr>
<td>85</td>
<td>3</td>
<td>MOV</td>
<td>data addr, data addr</td>
</tr>
<tr>
<td>86</td>
<td>2</td>
<td>MOV</td>
<td>data addr,@R0</td>
</tr>
<tr>
<td>87</td>
<td>2</td>
<td>MOV</td>
<td>data addr,@R1</td>
</tr>
<tr>
<td>88</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R0</td>
</tr>
<tr>
<td>89</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R1</td>
</tr>
<tr>
<td>8A</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R2</td>
</tr>
<tr>
<td>8B</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R3</td>
</tr>
<tr>
<td>8C</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R4</td>
</tr>
<tr>
<td>8D</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R5</td>
</tr>
<tr>
<td>8E</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R6</td>
</tr>
<tr>
<td>8F</td>
<td>2</td>
<td>MOV</td>
<td>data addr,R7</td>
</tr>
<tr>
<td>90</td>
<td>3</td>
<td>MOV</td>
<td>DPTR,#data</td>
</tr>
<tr>
<td>91</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>92</td>
<td>2</td>
<td>MOV</td>
<td>bit addr,C</td>
</tr>
<tr>
<td>93</td>
<td>1</td>
<td>MOVC</td>
<td>A,@A+DPTR</td>
</tr>
<tr>
<td>94</td>
<td>2</td>
<td>SUBB</td>
<td>A,#data</td>
</tr>
<tr>
<td>95</td>
<td>2</td>
<td>SUBB</td>
<td>A,data addr</td>
</tr>
<tr>
<td>96</td>
<td>1</td>
<td>SUBB</td>
<td>A,@R0</td>
</tr>
<tr>
<td>97</td>
<td>1</td>
<td>SUBB</td>
<td>A,@R1</td>
</tr>
<tr>
<td>98</td>
<td>1</td>
<td>SUBB</td>
<td>A,R0</td>
</tr>
<tr>
<td>99</td>
<td>1</td>
<td>SUBB</td>
<td>A,R1</td>
</tr>
<tr>
<td>9A</td>
<td>1</td>
<td>SUBB</td>
<td>A,R2</td>
</tr>
<tr>
<td>9B</td>
<td>1</td>
<td>SUBB</td>
<td>A,R3</td>
</tr>
<tr>
<td>9C</td>
<td>1</td>
<td>SUBB</td>
<td>A,R4</td>
</tr>
<tr>
<td>9D</td>
<td>1</td>
<td>SUBB</td>
<td>A,R5</td>
</tr>
<tr>
<td>9E</td>
<td>1</td>
<td>SUBB</td>
<td>A,R6</td>
</tr>
<tr>
<td>9F</td>
<td>1</td>
<td>SUBB</td>
<td>A,R7</td>
</tr>
<tr>
<td>A0</td>
<td>2</td>
<td>ORL</td>
<td>C, bit addr</td>
</tr>
<tr>
<td>A1</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>A2</td>
<td>2</td>
<td>MOV</td>
<td>C, bit addr</td>
</tr>
<tr>
<td>A3</td>
<td>1</td>
<td>INC</td>
<td>DPTR</td>
</tr>
<tr>
<td>A4</td>
<td>1</td>
<td>MUL</td>
<td>AB</td>
</tr>
<tr>
<td>A5</td>
<td>1</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>2</td>
<td>MOV</td>
<td>@R0, data addr</td>
</tr>
<tr>
<td>A7</td>
<td>2</td>
<td>MOV</td>
<td>@R1, data addr</td>
</tr>
<tr>
<td>A8</td>
<td>2</td>
<td>MOV</td>
<td>R0, data addr</td>
</tr>
<tr>
<td>A9</td>
<td>2</td>
<td>MOV</td>
<td>R1, data addr</td>
</tr>
<tr>
<td>AA</td>
<td>2</td>
<td>MOV</td>
<td>R2, data addr</td>
</tr>
<tr>
<td>AB</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>AC</td>
<td>2</td>
<td>MOV</td>
<td>R4, data addr</td>
</tr>
<tr>
<td>AD</td>
<td>2</td>
<td>MOV</td>
<td>R5, data addr</td>
</tr>
<tr>
<td>AE</td>
<td>2</td>
<td>MOV</td>
<td>R6, data addr</td>
</tr>
<tr>
<td>AF</td>
<td>2</td>
<td>MOV</td>
<td>R7, data addr</td>
</tr>
<tr>
<td>B0</td>
<td>2</td>
<td>ANL</td>
<td>C, bit addr</td>
</tr>
<tr>
<td>B1</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>B2</td>
<td>2</td>
<td>CPL</td>
<td>bit addr</td>
</tr>
<tr>
<td>B3</td>
<td>1</td>
<td>CPL</td>
<td>C</td>
</tr>
<tr>
<td>B4</td>
<td>3</td>
<td>CJNE</td>
<td>A, #data, code addr</td>
</tr>
<tr>
<td>B5</td>
<td>3</td>
<td>CJNE</td>
<td>A, data addr, code addr</td>
</tr>
<tr>
<td>B6</td>
<td>3</td>
<td>CJNE</td>
<td>@R0, #data, code addr</td>
</tr>
<tr>
<td>B7</td>
<td>3</td>
<td>CJNE</td>
<td>@R1, #data, code addr</td>
</tr>
<tr>
<td>B8</td>
<td>3</td>
<td>CJNE</td>
<td>R0, #data, code addr</td>
</tr>
<tr>
<td>B9</td>
<td>3</td>
<td>CJNE</td>
<td>R1, #data, code addr</td>
</tr>
<tr>
<td>BA</td>
<td>3</td>
<td>CJNE</td>
<td>R2, #data, code addr</td>
</tr>
<tr>
<td>BB</td>
<td>3</td>
<td>CJNE</td>
<td>R3, #data, code addr</td>
</tr>
<tr>
<td>BC</td>
<td>3</td>
<td>CJNE</td>
<td>R4, #data, code addr</td>
</tr>
<tr>
<td>BD</td>
<td>3</td>
<td>CJNE</td>
<td>R5, #data, code addr</td>
</tr>
<tr>
<td>BE</td>
<td>3</td>
<td>CJNE</td>
<td>R6, #data, code addr</td>
</tr>
<tr>
<td>BF</td>
<td>3</td>
<td>CJNE</td>
<td>R7, #data, code addr</td>
</tr>
<tr>
<td>C0</td>
<td>2</td>
<td>PUSH</td>
<td>data addr</td>
</tr>
<tr>
<td>C1</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>C2</td>
<td>2</td>
<td>CLR</td>
<td>bit addr</td>
</tr>
<tr>
<td>C3</td>
<td>1</td>
<td>CLR</td>
<td>C</td>
</tr>
<tr>
<td>C4</td>
<td>1</td>
<td>SWAP</td>
<td>A</td>
</tr>
<tr>
<td>C5</td>
<td>2</td>
<td>XCH</td>
<td>A, data addr</td>
</tr>
<tr>
<td>C6</td>
<td>1</td>
<td>XCH</td>
<td>A, @R0</td>
</tr>
<tr>
<td>C7</td>
<td>1</td>
<td>XCH</td>
<td>A, @R1</td>
</tr>
<tr>
<td>C8</td>
<td>1</td>
<td>XCH</td>
<td>A, R0</td>
</tr>
<tr>
<td>C9</td>
<td>1</td>
<td>XCH</td>
<td>A, R1</td>
</tr>
<tr>
<td>CA</td>
<td>1</td>
<td>XCH</td>
<td>A, R2</td>
</tr>
<tr>
<td>CB</td>
<td>1</td>
<td>XCH</td>
<td>A, R3</td>
</tr>
</tbody>
</table>
Table 3. Instruction Opcodes in Hexadecimal Order (Continued)

<table>
<thead>
<tr>
<th>Hex Code</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>1</td>
<td>XCH</td>
<td>A,R4</td>
</tr>
<tr>
<td>CD</td>
<td>1</td>
<td>XCH</td>
<td>A,R5</td>
</tr>
<tr>
<td>CE</td>
<td>1</td>
<td>XCH</td>
<td>A,R6</td>
</tr>
<tr>
<td>CF</td>
<td>1</td>
<td>XCH</td>
<td>A,R7</td>
</tr>
<tr>
<td>D0</td>
<td>2</td>
<td>POP</td>
<td>data addr</td>
</tr>
<tr>
<td>D1</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>D2</td>
<td>2</td>
<td>SETB</td>
<td>bit addr</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>SETB</td>
<td>C</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>DA</td>
<td>A</td>
</tr>
<tr>
<td>D5</td>
<td>3</td>
<td>DJNZ</td>
<td>data addr,code addr</td>
</tr>
<tr>
<td>D6</td>
<td>1</td>
<td>XCHD</td>
<td>A,R0</td>
</tr>
<tr>
<td>D7</td>
<td>1</td>
<td>XCHD</td>
<td>A,R1</td>
</tr>
<tr>
<td>D8</td>
<td>2</td>
<td>DJNZ</td>
<td>R0,code addr</td>
</tr>
<tr>
<td>D9</td>
<td>2</td>
<td>DJNZ</td>
<td>R1,code addr</td>
</tr>
<tr>
<td>DA</td>
<td>2</td>
<td>DJNZ</td>
<td>R2,code addr</td>
</tr>
<tr>
<td>DB</td>
<td>2</td>
<td>DJNZ</td>
<td>R3,code addr</td>
</tr>
<tr>
<td>DC</td>
<td>2</td>
<td>DJNZ</td>
<td>R4,code addr</td>
</tr>
<tr>
<td>DD</td>
<td>2</td>
<td>DJNZ</td>
<td>R5,code addr</td>
</tr>
<tr>
<td>DE</td>
<td>2</td>
<td>DJNZ</td>
<td>R6,code addr</td>
</tr>
<tr>
<td>DF</td>
<td>2</td>
<td>DJNZ</td>
<td>R7,code addr</td>
</tr>
<tr>
<td>E0</td>
<td>1</td>
<td>MOVX</td>
<td>A,@DPTR</td>
</tr>
<tr>
<td>E1</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>E2</td>
<td>1</td>
<td>MOVX</td>
<td>A,@R0</td>
</tr>
<tr>
<td>E3</td>
<td>1</td>
<td>MOVX</td>
<td>A,@R1</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>CLR</td>
<td>A</td>
</tr>
<tr>
<td>E5</td>
<td>2</td>
<td>MOV</td>
<td>A,data addr</td>
</tr>
<tr>
<td>E6</td>
<td>1</td>
<td>MOV</td>
<td>A,@R0</td>
</tr>
<tr>
<td>E7</td>
<td>1</td>
<td>MOV</td>
<td>A,@R1</td>
</tr>
<tr>
<td>E8</td>
<td>1</td>
<td>MOV</td>
<td>A,R0</td>
</tr>
<tr>
<td>E9</td>
<td>1</td>
<td>MOV</td>
<td>A,R1</td>
</tr>
<tr>
<td>EA</td>
<td>1</td>
<td>MOV</td>
<td>A,R2</td>
</tr>
<tr>
<td>EB</td>
<td>1</td>
<td>MOV</td>
<td>A,R3</td>
</tr>
<tr>
<td>EC</td>
<td>1</td>
<td>MOV</td>
<td>A,R4</td>
</tr>
<tr>
<td>ED</td>
<td>1</td>
<td>MOV</td>
<td>A,R5</td>
</tr>
<tr>
<td>EE</td>
<td>1</td>
<td>MOV</td>
<td>A,R6</td>
</tr>
<tr>
<td>EF</td>
<td>1</td>
<td>MOV</td>
<td>A,R7</td>
</tr>
<tr>
<td>F0</td>
<td>1</td>
<td>MOVX</td>
<td>@DPTR,A</td>
</tr>
<tr>
<td>F1</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>MOVX</td>
<td>@R0,A</td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>MOVX</td>
<td>@R1,A</td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>CPL</td>
<td>A</td>
</tr>
<tr>
<td>F5</td>
<td>2</td>
<td>MOV</td>
<td>data addr,A</td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>MOV</td>
<td>@R0,A</td>
</tr>
<tr>
<td>F7</td>
<td>1</td>
<td>MOV</td>
<td>@R1,A</td>
</tr>
<tr>
<td>F8</td>
<td>1</td>
<td>MOV</td>
<td>R0,A</td>
</tr>
<tr>
<td>F9</td>
<td>1</td>
<td>MOV</td>
<td>R1,A</td>
</tr>
<tr>
<td>FA</td>
<td>1</td>
<td>MOV</td>
<td>R2,A</td>
</tr>
<tr>
<td>FB</td>
<td>1</td>
<td>MOV</td>
<td>R3,A</td>
</tr>
<tr>
<td>FC</td>
<td>1</td>
<td>MOV</td>
<td>R4,A</td>
</tr>
<tr>
<td>FD</td>
<td>1</td>
<td>MOV</td>
<td>R5,A</td>
</tr>
<tr>
<td>FE</td>
<td>1</td>
<td>MOV</td>
<td>R6,A</td>
</tr>
<tr>
<td>FF</td>
<td>1</td>
<td>MOV</td>
<td>R7,A</td>
</tr>
</tbody>
</table>
The Intel EXPRESS system offers enhancements to the operational specifications of the MCS®-51 family of microcontrollers. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards.

The EXPRESS program includes the commercial standard temperature range with burn-in, and an extended temperature range with or without burn-in.

With the commercial standard temperature range operational characteristics are guaranteed over the temperature range of 0°C to 70°C. With the extended temperature range option, operational characteristics are guaranteed over the range of −40°C to +85°C.

The optional burn-in is dynamic, for a minimum time of 160 hours at 125°C with $V_{CC} = 5.5V \pm 0.5V$, following guidelines in MIL-STD-883, Method 1015.

Package types and EXPRESS versions are identified by a one- or two-letter prefix to the part number. The prefixes are listed in Table 1.

For the extended temperature range option, this data sheet specifies the parameters which deviate from their commercial temperature range limits. The commercial temperature range data sheets are applicable for all parameters not listed here.

### Electrical Deviations from Commercial Specifications for Extended Temperature Range

D.C. and A.C. parameters not included here are the same as in the commercial temperature range data sheets.

#### D.C. CHARACTERISTICS: ($T_A = -40°C$ to $+85°C; V_{CC} = 5V \pm 10%; V_{SS} = 0V$)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IL}$</td>
<td>Input Low Voltage</td>
<td>−0.5</td>
<td>0.75</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>Input High Voltage (Except XTAL2, RST)</td>
<td>2.1</td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>$I_{CC}$</td>
<td>Power Supply Current: 8051AH,8031AH 8052AH,8032AH 8751H,8751H</td>
<td>135</td>
<td>ma</td>
<td>All Outputs Disconnected; $EA = V_{CC}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>175</td>
<td>ma</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>265</td>
<td>ma</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 1 — Prefix Identification

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Package Type</th>
<th>Temperature Range</th>
<th>Burn-In</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>plastic</td>
<td>commercial</td>
<td>no</td>
</tr>
<tr>
<td>D</td>
<td>cerdip</td>
<td>commercial</td>
<td>no</td>
</tr>
<tr>
<td>C</td>
<td>ceramic</td>
<td>commercial</td>
<td>no</td>
</tr>
<tr>
<td>TP</td>
<td>plastic</td>
<td>extended</td>
<td>no</td>
</tr>
<tr>
<td>TD</td>
<td>cerdip</td>
<td>extended</td>
<td>no</td>
</tr>
<tr>
<td>TC</td>
<td>ceramic</td>
<td>extended</td>
<td>no</td>
</tr>
<tr>
<td>QP</td>
<td>plastic</td>
<td>commercial</td>
<td>yes</td>
</tr>
<tr>
<td>QD</td>
<td>cerdip</td>
<td>commercial</td>
<td>yes</td>
</tr>
<tr>
<td>QC</td>
<td>ceramic</td>
<td>commercial</td>
<td>yes</td>
</tr>
<tr>
<td>LP</td>
<td>plastic</td>
<td>extended</td>
<td>yes</td>
</tr>
<tr>
<td>LD</td>
<td>cerdip</td>
<td>extended</td>
<td>yes</td>
</tr>
<tr>
<td>LC</td>
<td>ceramic</td>
<td>extended</td>
<td>yes</td>
</tr>
</tbody>
</table>

Please note:
- Commercial temperature range is 0° to 70°C. Extended temperature range is −40° to +85°C.
- Burn-in is dynamic, for a minimum time of 160 hours at 125°C, \( V_{CC} = 5.5V \pm 0.5V \), following guidelines in MIL-STD-883 Method 1015 (Test Condition D).
- The following devices are not available in plastic packages: 8751 H.
- The following devices are not available in ceramic packages: 8051AH, 8031AH, 8052AH, 8032AH.

Examples: P8031AH indicates 8031AH in a plastic package and specified for commercial temperature range, without burn-in. LD8751H indicates 8751H in a cerdip package and specified for extended temperature range with burn-in.
An Introduction to the Intel MCS®-51 Single-Chip Microcomputer Family

Contents

1. INTRODUCTION ........................................ 10-2
   Family Overview ........................................ 10-2
   Microcomputer Background Concepts ....................... 10-3

2. ARCHITECTURE AND ORGANIZATION ..................... 10-5
   Central Processing Unit .................................. 10-6
   Memory Spaces .......................................... 10-9
   Input/Output Ports ...................................... 10-10
   Special Peripheral Functions ............................ 10-11

3. INSTRUCTION SET AND ADDRESSING MODES ............. 10-15
   Data Addressing Modes ................................... 10-15
   Addressing Mode Combinations .......................... 10-18
   Advantages of Symbolic Addressing ...................... 10-18
   Arithmetic Instruction Usage ............................ 10-19
   Multiplication and Division ............................. 10-20
   Logical Byte Operations ................................ 10-20
   Program Control ........................................ 10-21
   Operate-and-Branch Instructions ....................... 10-22
   Stack Operations ....................................... 10-22
   Table Look-Up Instructions .............................. 10-23

4. BOOLEAN PROCESSING INSTRUCTIONS .................. 10-25
   Direct Bit Addressing ................................... 10-25
   Bit Manipulation Instructions ......................... 10-25
   Solving Combinatorial Logic Equations .................. 10-26

5. ON-CHIP PERIPHERAL FUNCTIONS ...................... 10-28
   I/O Ports .............................................. 10-28
   Serial Port and Timer .................................. 10-29

6. SUMMARY .................................................. 10-30
1. INTRODUCTION

In 1976 Intel introduced the MCS-48™ family, consisting of the 8048, 8748, and 8035 microcomputers. These parts marked the first time a complete microcomputer system, including an eight-bit CPU, 1024 8-bit words of ROM or EPROM program memory, 64 words of data memory, I/O ports and an eight-bit timer/counter could be integrated onto a single silicon chip. Depending only on the program memory contents, one chip could control a limitless variety of products, ranging from appliances or automobile engines to text or data processing equipment. Follow-on products stretched the MCS-48™ architecture in several directions: the 8049 and 8039 doubled the amount of on-chip memory and ran 83% faster; the 8021 reduced costs by executing a subset of the 8048 instructions with a somewhat slower clock; and the 8022 put a unique two-channel 8-bit analog-to-digital converter on the same NMOS chip as the computer, letting the chip interface directly with analog transducers.

Now three new high-performance single-chip microcomputers—the Intel® 8051, 8751, and 8031—extend the advantages of Integrated Electronics to whole new product areas. Thanks to Intel's new HMOS technology, the MCS-51™ family provides four times the program memory and twice the data memory as the 8048 on a single chip. New I/O and peripheral capabilities both increase the range of applicability and reduce total system cost. Depending on the use, processing throughput increases by two and one-half to ten times.

This Application Note is intended to introduce the reader to the MCS-51™ architecture and features. While it does not assume intimacy with the MCS-48™ product line on the part of the reader, he/she should be familiar with some microprocessor (preferably Intel's, of course) or have a background in computer programming and digital logic.

Family Overview

Pinout diagrams for the 8051, 8751, and 8031 are shown in Figure 1. The devices include the following features:

- Single-supply 5 volt operation using HMOS technology.
- 4096 bytes program memory on-chip (not on 8031).
- 128 bytes data memory on-chip.
- Four register banks.
- 128 User-defined software flags.
- 64 Kilobytes each program and external RAM addressability.
- One microsecond instruction cycle with 12 MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
- Multiple mode, high-speed programmable Serial Port.
- Two multiple mode, 16-bit Timer/Counters.
- Two-level prioritized interrupt structure.
- Full depth stack for subroutine return linkage and data storage.
- Augmented MCS-48™ instruction set.
- Direct Byte and Bit addressability.
- Binary or Decimal arithmetic.
- Signed-overflow detection and parity computation.
- Hardware Multiple and Divide in 4 usec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.
All three devices come in a standard 40-pin Dual In-Line Package, with the same pin-out, the same timing, and the same electrical characteristics. The primary difference between the three is the on-chip program memory—different types are offered to satisfy differing user requirements.

The 8751 provides 4K bytes of ultraviolet-Erasable, Programmable Read Only Memory (EPROM) for program development, prototyping, and limited production runs. (By convention, 1K means $2^{10} = 1024$. 1k—with a lower case “k”—equals $10^3 = 1000$.) This part may be individually programmed for a specific application using Intel's Universal PROM Programmer (UPP). If software bugs are detected or design specifications change from the same part may be "erased" in a matter of minutes by exposure to ultraviolet light and reprogrammed with the modified code. This cycle may be repeated indefinitely during the design and development phase.

The final version of the software must be programmed into a large number of production parts. The 8051 has 4K bytes of ROM which are mask-programmed with the customer's order when the chip is built. This part is considerably less expensive, but cannot be erased or altered after fabrication.

The 8031 does not have any program memory on-chip, but may be used with up to 64K bytes of external standard or multiplexed ROMs, PROMs, or EPROMs. The 8031 fits well in applications requiring significantly larger or smaller amounts of memory than the 4K bytes provided by its two siblings.

(The 8051 and 8751 automatically access external program memory for all addresses greater than the 4096 bytes on-chip. The External Access input is an override for all internal program memory—the 8051 and 8751 will each emulate an 8031 when pin 31 is low.)

Throughout this Note, "8051" is used as a generic term. Unless specifically stated otherwise, the point applies equally to all three components. Table I summarizes the quantitative differences between the members of the MCS-48™ and MCS-51™ families.

The remainder of this Note discusses the various MCS-51™ features and how they are used. Software and/or hard-

<table>
<thead>
<tr>
<th>Table 1. Features of Intel's Single-Chip Microcomputers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPROM Program Memory</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>8021</td>
</tr>
<tr>
<td>8748</td>
</tr>
<tr>
<td>8751</td>
</tr>
</tbody>
</table>
The letters “MCS” have traditionally indicated a system or family of compatible Intel® microcomputer components, including CPUs, memories, clock generators, I/O expanders, and so forth. The numerical suffix indicates the microprocessor or microcomputer which serves as the cornerstone of the family. Microcomputers in the MCS-48™ family currently include the 8048-series (8035, 8048, & 8748), the 8049-series (8039 & 8049), and the 8021 and 8022; the family also includes the 8243, an I/O expander compatible with each of the microcomputers. Each computer’s CPU is derived from the 8048, with essentially the same architecture, addressing modes, and instruction set, and a single assembler (ASM48) serves each.

The first members of the MCS-51™ family are the 8051, 8751, and 8031. The architecture of the 8051-series, while derived from the 8048, is not strictly compatible; there are more addressing modes, more instructions, larger address spaces, and a few other hardware differences. In this Application Note the letters “MCS-51” are used when referring to architectural features of the 8051-series—features which would be included on possible future microcomputers based on the 8051 CPU. Such products could have different amounts of memory (as in the 8048/8049) or different peripheral functions (as in the 8021 and 8022) while leaving the CPU and instruction set intact. ASM51 is the assembler used by all microcomputers in the 8051 family.

Two digit decimal numbers may be “packed” in an eight-bit value, using four bits for the binary code of each digit. This is called Binary-Coded Decimal (BCD) representation, and is often used internally in programs which interact heavily with human beings.

Alphanumeric characters (letters, numbers, punctuation marks, etc.) are often represented using the American Standard Code for Information Interchange (ASCII) convention. Each character is associated with a unique seven-bit binary number. Thus one byte may represent a single character, and a word or sequence of letters may be represented by a series (or “string”) of bytes. Since the ASCII code only uses 128 characters, the most significant bit of the byte is not needed to distinguish between characters. Often D7 is set to 0 for all characters. In some coding schemes, D7 is used to indicate the “parity” of the other seven bits—set or cleared as necessary to ensure that the total number of “1” bits in the eight-bit code is even (“even parity”) or odd (“odd parity”). The 8051 includes hardware to compute parity when it is needed.

A computer program consists of an ordered sequence of specific, simple steps to be executed by the CPU one-at-a-time. The method or sequence of steps used collectively to solve the user’s application is called an “algorithm.”

The program is stored inside the computer as a sequence of binary numbers, where each number corresponds to one of the basic operations (“opcodes”) which the CPU is capable of executing. In the 8051, each program memory location is one byte. A complete instruction consists of a sequence of one or more bytes, where the first defines the operation to be executed and additional bytes (if needed) hold additional information, such as data values or variable addresses. No instruction is longer than three bytes.

The way in which binary opcodes and modifier bytes are assigned to the CPU’s operations is called the computer’s “machine language.” Writing a program directly in machine language is time-consuming and tedious. Human beings think in words and concepts rather than encoded numbers, so each CPU operation and resource is given a name and standard abbreviation (“mnemonic”). Programs are more easily discussed using these standard mnemonics, or “assembly language,” and may be typed into an Intel® Intellec® 800 or Series II® microcomputer development system in this form. The development system can mechanically translate the program from assembly language “source” form to machine language “object” code using a program called an “assembler.” The MCS-51™ assembler is called ASM51.

There are several important differences between a computer’s machine language and the assembly language used as a tool to represent it. The machine language or instruction set is the set of operations which the CPU can perform while a program is executing (“at run-time”), and is strictly determined by the microcomputer hardware design.

The assembly language is a standard (though more-or-less arbitrary) set of symbols including the instruction set mnemonics, but with additional features which further simplify the program design process. For example, ASM51 has controls for creating and formatting a program listing, and a number of directives for allocating variable storage and inserting arbitrary bytes of data into the object code for creating tables of constants.
In addition, ASM51 can perform sophisticated mathematical operations, computing addresses or evaluating arithmetic expressions to relieve the programmer from this drudgery. However, these calculations can only use information known at “assembly time.”

For example, the 8051 performs arithmetic calculations at run-time, eight bits at a time. ASM51 can do similar operations 16 bits at a time. The 8051 can only do one simple step per instruction, while ASM51 can perform complex calculations in each line of source code. However, the operations performed by the assembler may only use parameter values fixed at assembly-time, not variables whose values are unknown until program execution begins.

For example, when the assembly language source line,

```
ADD A, # (LOOP_COUNT + 1) * 3
```

is assembled, ASM51 will find the value of the previously-defined constant “LOOP_COUNT” in an internal symbol table, increment the value, multiply the sum by three, and (assuming it is between -256 and 255 inclusive) truncate the product to eight bits. When this instruction is executed, the 8051 ALU will just add that resulting constant to the accumulator.

Some similar differences exist to distinguish number system (“radix”) specifications. The 8051 does all computations in binary (though there are provisions for then converting the result to decimal form). In the course of writing a program, though, it may be more convenient to specify constants using some other radix, such as base 10. On other occasions, it is desirable to specify the ASCII code for some character or string of characters without referring to tables. ASM51 allows several representations for constants, which are converted to binary as each instruction is assembled.

For example, binary numbers are represented in the assembly language by a series of ones and zeros (naturally), followed by the letter “B” (for Binary); octal numbers as a series of octal digits (0-7) followed by the letter “O” (for Octal) or “Q” (which doesn’t stand for anything, but looks sort of like an “O” and is less likely to be confused with a zero).

Hexadecimal numbers are represented by a series of hexadecimal digits (0-9, A-F), followed by (you guessed it) the letter “H.” A “hex” number must begin with a decimal digit; otherwise it would look like a user-defined symbol (to be discussed later). A “dummy” leading zero may be inserted before the first digit to meet this constraint. The character string “BACH” could be a legal label for a Baroque music synthesis routine; the string “0BACH” is the hexadecimal constant $BAC_{16}$. This is a case where adding 0 makes a big difference.

Decimal numbers are represented by a sequence of decimal digits, optionally followed by a “.D.” If a number has no suffix, it is assumed to be decimal—so it had better not contain any non-decimal digits. “0BAC” is not a legal representation for anything.

When an ASCII code is needed in a program, enclose the desired character between two apostrophes (as in ’#”) and the assembler will convert it to the appropriate code (in this case 23H). A string of characters between apostrophes is translated into a series of constants; ‘BACH’ becomes 42H, 41H, 43H, 48H.

These same conventions are used throughout the associated Intel documentation. Table 2 illustrates some of the different number formats.

### 2. ARCHITECTURE AND ORGANIZATION

Figure 3 blocks out the MCS-51™ internal organization. Each microcomputer combines a Central Processing Unit, two kinds of memory (data RAM plus program ROM or EPROM), Input/Output ports, and the mode,

<table>
<thead>
<tr>
<th>Bit Pattern</th>
<th>Binary</th>
<th>Octal</th>
<th>Hexa-Decimal</th>
<th>Decimal</th>
<th>Signed Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0B</td>
<td>0Q</td>
<td>00H</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1</td>
<td>1B</td>
<td>1Q</td>
<td>01H</td>
<td>1</td>
<td>+1</td>
</tr>
<tr>
<td>0 0 0 0 0 1 1 1</td>
<td>111B</td>
<td>7Q</td>
<td>07H</td>
<td>7</td>
<td>+7</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0 0</td>
<td>1000B</td>
<td>10Q</td>
<td>08H</td>
<td>8</td>
<td>+8</td>
</tr>
<tr>
<td>0 0 0 0 1 0 0 1</td>
<td>1001B</td>
<td>11Q</td>
<td>09H</td>
<td>9</td>
<td>+9</td>
</tr>
<tr>
<td>0 0 0 1 0 1 0</td>
<td>1010B</td>
<td>12Q</td>
<td>0AH</td>
<td>10</td>
<td>+10</td>
</tr>
<tr>
<td>0 0 0 1 1 1 1</td>
<td>1111B</td>
<td>17Q</td>
<td>0FH</td>
<td>15</td>
<td>+15</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>1000B</td>
<td>20Q</td>
<td>10H</td>
<td>16</td>
<td>+16</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1</td>
<td>111111B</td>
<td>177Q</td>
<td>7FH</td>
<td>127</td>
<td>+127</td>
</tr>
<tr>
<td>1 0 0 0 0 0 0</td>
<td>1000000B</td>
<td>200Q</td>
<td>80H</td>
<td>128</td>
<td>-128</td>
</tr>
<tr>
<td>1 0 0 0 0 0 1</td>
<td>1000001B</td>
<td>201Q</td>
<td>81H</td>
<td>129</td>
<td>-127</td>
</tr>
<tr>
<td>1 1 1 1 1 1 0</td>
<td>1111110B</td>
<td>376Q</td>
<td>0FEH</td>
<td>254</td>
<td>-2</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1</td>
<td>1111111B</td>
<td>377Q</td>
<td>0FFH</td>
<td>255</td>
<td>-1</td>
</tr>
</tbody>
</table>
status, and data registers and random logic needed for a variety of peripheral functions. These elements communicate through an eight-bit data bus which runs throughout the chip, somewhat akin to indoor plumbing. This bus is buffered to the outside world through an I/O port when memory or I/O expansion is desired.

Let's summarize what each block does; later chapters dig into the CPU's instruction set and the peripheral registers in much greater detail.

**Central Processing Unit**
The CPU is the "brains" of the microcomputer, reading the user's program and executing the instructions stored therein. Its primary elements are an eight-bit Arithmetic/Logic Unit with associated registers A, B, PSW, and SP, and the sixteen-bit Program Counter and "Data Pointer" registers.
**Arithmetic Logic Unit**

The ALU can perform (as the name implies) arithmetic and logic functions on eight-bit variables. The former include basic addition, subtraction, multiplication, and division; the latter include the logical operations AND, OR, and Exclusive-OR, as well as rotate, clear, complement, and so forth. The ALU also makes conditional branching decisions, and provides data paths and temporary registers used for data transfers within the system. Other instructions are built up from these primitive functions: the addition capability can increment registers or automatically compute program destination addresses; subtraction is also used in decrementing or comparing the magnitude of two variables.

These primitive operations are automatically cascaded and combined with dedicated logic to build complex instructions such as incrementing a sixteen-bit register pair. To execute one form of the compare instruction, for example, the 8051 increments the program counter three times, reads three bytes of program memory, computes a register address with logical operations, reads internal data memory twice, makes an arithmetic comparison of two variables, computes a sixteen-bit destination address, and decides whether or not to make a branch—all in two microseconds!

An important and unique feature of the MCS-51 architecture is that the ALU can also manipulate one-bit as well as eight-bit data types. Individual bits may be set, cleared, or complemented, moved, tested, and used in logic computations. While support for a more primitive data type may initially seem a step backwards in an era of increasing word length, it makes the 8051 especially well suited for controller-type applications. Such algorithms inherently involve Boolean (true/false) input and output variables, which were heretofore difficult to implement with standard microprocessors. These features are collectively referred to as the MCS-51™ “Boolean Processor,” and are described in the so-named chapter to come.

Thanks to this powerful ALU, the 8051 instruction set fares well at both real-time control and data intensive algorithms. A total of 51 separate operations move and manipulate three data types: Boolean (1-bit), byte (8-bit), and address (16-bit). All told, there are eleven addressing modes—seven for data, four for program sequence control (though only eight are used by more than just a few specialized instructions). Most operations allow several addressing modes, bringing the total number of instructions (operation/addressing mode combinations) to 111, encompassing 255 of the 256 possible eight-bit instruction opcodes.

### Instruction Set Overview

Table 4 lists these 111 instructions classified into five groups:

- Arithmetic Operations
- Logical Operations for Byte Variables
- Data Transfer Instructions
- Boolean Variable Manipulation
- Program Branching and Machine Control

MCS-48™ programmers perusing Table 4 will notice the absence of special categories for Input/Output, Timer/Counter, or Control instructions. These functions are all still provided (and indeed many new functions are added), but as special cases of more generalized operations in other categories. To explicitly list all the useful instructions involving I/O and peripheral registers would require a table approximately four times as long.

Observant readers will also notice that all of the 8048's page-oriented instructions (conditional jumps, JMPP, MOV P, MOV P3) have been replaced with corresponding but non-paged instructions. The 8051 instruction set is entirely non-page-oriented. The MCS-48™ “MOV P” instruction replacement and all conditional jump instructions operate relative to the program counter, with the actual jump address computed by the CPU during instruction execution. The “MOV P3” and “JMPP” replacements are now made relative to another sixteen-bit register, which allows the effective destination to be anywhere in the program memory space, regardless of where the instruction itself is located. There are even three-byte jump and call instructions allowing the destination to be anywhere in the 64K program address space.

The instruction set is designed to make programs efficient both in terms of code size and execution speed. No instruction requires more than three bytes of program memory, with the majority requiring only one or two bytes. Virtually all instructions execute in either one or two instruction cycles—one or two microseconds with a 12-MHz crystal—with the sole exceptions (multiply and divide) completing in four cycles.

Many instructions such as arithmetic and logical functions or program control, provide both a short and a long form for the same operation, allowing the programmer to optimize the code produced for a specific application. The 8051 usually fetches two instruction bytes per instruction cycle, so using a shorter form can lead to faster execution as well.

For example, any byte of RAM may be loaded with a constant with a three-byte, two-cycle instruction, but the commonly used “working registers” in RAM may be initialized in one cycle with a two-byte form. Any bit anywhere on the chip may be set, cleared, or complemented by a single three-byte logical instruction using two cycles. But critical control bits, I/O pins, and software flags may be controlled by two-byte, single cycle instructions. While three-byte jumps and calls can go anywhere in program memory, nearby sections of code may be reached by shorter relative or absolute versions.
Accumulator and software. Generalized addressing modes for byte and bit operations can be the source or destination for logical operations including table look-ups and external RAM expansion. Several functions apply exclusively to the accumulator: rotates, parity computation, testing for zero, and so on.

Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word shown in Figure 4.

The period within entries under the Position column is called the "dot operator," and indicates a particular bit position within an eight-bit byte. "PSW.5" specifies bit 5 of the PSW. Both the documentation and ASM51 use this notation.

The most "active" status bit is called the carry flag (abbreviated "C"). This bit makes possible multiple precision arithmetic operations including addition, subtraction, and rotates. The carry also serves as a "Boolean accumulator" for one-bit logical operations and bit manipulation instructions. The overflow flag (OV) detects when arithmetic overflow occurs on signed integer operands, making two's complement arithmetic possible. The parity flag (P) is updated after every instruction cycle with the even-parity of the accumulator contents.

The CPU does not control the two-register-bank select bits, RS1 and RS0. Rather, they are manipulated by software to enable one of the four register banks. The usage of the PSW flags is demonstrated in the Instruction Set chapter of this Note.

Even though the architecture is accumulator-based, provisions have been made to bypass the accumulator in common instruction situations. Data may be moved from any location on-chip to any register, address, or indirect address (and vice versa), any register may be loaded with a constant, etc., all without affecting the accumulator. Logical operations may be performed against registers or variables to alter fields of bits—without using or affecting the accumulator. Variables may be incremented, decremented, or tested without using the accumulator. Flags and control bits may be manipulated and tested without affecting anything else.

Other CPU Registers
A special eight-bit register ("B") serves in the execution of the multiply and divide instructions. This register is used in conjunction with the accumulator as the second input operand and to return eight-bits of the result.

The MCS-51 family processors include a hardware stack within internal RAM, useful for subroutine linkage,
passing parameters between routines, temporary variable storage, or saving status during interrupt service routines. The Stack Pointer (SP) is an eight-bit pointer register which indicates the address of the last byte pushed onto the stack. The stack pointer is automatically incremented or decremented on all push or pop instructions and all subroutine calls and returns. In theory, the stack in the 8051 may be up to a full 128 bytes deep. (In practice, even simple programs would use a handful of RAM locations for pointers, variables, and so forth—reducing the stack depth by that number.) The stack pointer defaults to 7 on reset, so that the stack will start growing up from location 8, just like in the 8048. By altering the pointer contents the stack may be relocated anywhere within internal RAM.

Finally, a 16-bit register called the data pointer (DPTR) serves as a base register in indirect jumps, table look-up instructions, and external data transfers. The high- and low-order halves of the data pointer may be manipulated as separate registers (DPH and DPL, respectively) or together using special instructions to load or increment all sixteen bits. Unlike the 8048, look-up tables can therefore start anywhere in program memory and be of arbitrary length.

Memory Spaces

Program memory is separate and distinct from data memory. Each memory type has a different addressing mechanism, different control signals, and a different function.

The program memory array (ROM or EPROM), like an elephant, is extremely large and never forgets information, even when power is removed. Program memory is used for information needed each time power is applied: initialization values, calibration constants, keyboard layout tables, etc., as well as the program itself. The program memory has a sixteen-bit address bus; its elements are addressed using the Program Counter or instructions which generate a sixteen-bit address.

To stretch our analogy just a bit, data memory is like a mouse: it is smaller and therefore quicker than program memory, and it goes into a random state when electrical power is applied. On-chip data RAM is used for variables which are determined or may change while the program is running.

A computer spends most of its time manipulating variables, not constants, and a relatively small number of variables at that. Since eight-bits is more than sufficient to uniquely address 128 RAM locations, the on-chip RAM address register is only one byte wide. In contrast to the program memory, data memory accesses need a single eight-bit value—a constant or another variable—to specify a unique location. Since this is the basic width of the ALU and the different memory types, those resources can be used by the addressing mechanisms, contributing greatly to the computer's operating efficiency.

The partitioning of program and data memory is extended to off-chip memory expansion. Each may be added independently, and each uses the same address and data busses, but with different control signals. External program memory is gated onto the external data bus by the PSEN (Program Store Enable) control output, pin 29. External data memory is read onto the bus by the RD output, pin 17, and written with data supplied from the microcomputer by the WR output, pin 16. (There is no control pin to write external program ROM, which is by definition Read Only.) While both types may be expanded to up to 64K bytes, the external data memory may optionally be expanded in 256 byte "pages" to preserve the use of P2 as an I/O port. This is useful with a relatively small expansion RAM (such as the Intel® 8155) or for addressing external peripherals.

Single-chip controller programs are finalized during the project design cycle, and are not modified after production. Intel's single-chip microcomputers are not "von Neumann" architectures common among main-frame and mini-computer systems: the MCS-51™ processor data memory—on-chip and external—may not be used for program code. Just as there is no write-control signal for program memory, there is no way for the CPU to execute instructions out of RAM. In return, this concession allows an architecture optimized for efficient controller applications: a large, fixed program located in ROM, a hundred or so variables in RAM, and different methods for efficiently addressing each.

(Von Neumann machines are helpful for software development and debug. An 8051 system could be modified to have a single off-chip memory space by gating together the two memory-read controls (PSEN and RD) with a two-input AND gate (Figure 5). The CPU could then write data into the common memory array using WR and
external data transfer instructions, and read instructions or data with the AND gate output and data transfer or program memory look-up instructions.)

In addition to the memory arrays, there is (yet) another (albeit sparsely populated) physical address space. Connected to the internal data bus are a score of special-purpose eight-bit registers scattered throughout the chip. Some of these—B, SP, PSW, DPH, and DPL—have been discussed above. Others—I/O ports and peripheral function registers—will be introduced in the following sections. Collectively, these registers are designated as the "special-function register" address space. Even the accumulator is assigned a spot in the special-function register address space for additional flexibility and uniformity.

Thus, the MCS-51™ architecture supports several distinct "physical" address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write control signals, or both:

- On-chip program memory;
- On-chip data memory;
- Off-chip program memory;
- Off-chip data memory;
- On-chip special-function registers.

What the programmer sees, though, are "logical" address spaces. For example, as far as the programmer is concerned, there is only one type of program memory, 64K bytes in length. The fact that it is formed by combining on- and off-chip arrays (split 4K/60K on the 8051 and 8751) is "invisible" to the programmer; the CPU automatically fetches each byte from the appropriate array, based on its address.

(Presumably, future microcomputers based on the MCS-51™ architecture may have a different physical split, with more or less of the 64K total implemented on-chip. Using the MCS-48™ family as a precedent, the 8048's 4K potential program address space was split 1K/3K between on- and off-chip arrays; the 8049's was split 2K/2K.)

Why go into such tedious details about address spaces? The logical addressing modes are described in the Instruction Set chapter in terms of physical address spaces. Understanding their differences now will pay off in understanding and using the chips later.

Input/Output Ports

The MCS-51™ I/O port structure is extremely versatile. The 8051 and 8751 each have 32 I/O pins configured as four eight-bit parallel ports (PO, P1, P2, and P3). Each pin will input or output data (or both) under software control, and each may be referenced by a wide repertoire of byte and bit operations.

In various operating or expansion modes, some of these I/O pins are also used for special input or output functions. Instructions which access external memory use Port 0 as a multiplexed address/data bus: at the beginning of an external memory cycle eight bits of the address are output on PO; later data is transferred on the same eight pins. External data transfer instructions which supply a sixteen-bit address, and any instruction accessing external program memory, output the high-order eight bits on P2 during the access cycle. (The 8031 always uses the pins of P0 and P2 for external addressing, but P1 and P3 are available for standard I/O.)

The eight pins of Port 3 (P3) each have a special function. Two external interrupts, two counter inputs, two serial data lines, and two timing control strobes use pins of P3 as described in Figure 6. Port 3 pins corresponding to functions not used are available for conventional I/O.

Even within a single port, I/O functions may be combined in many ways: input and output may be performed using different pins at the same time, or the same pins at different times; in parallel in some cases, and in serial in others; as test pins, or (in the case of Port 3) as additional special functions.
### Special Peripheral Functions

There are a few special needs common among control-oriented computer systems:

- keeping track of elapsed real-time;
- maintaining a count of signal transitions;
- measuring the precise width of input pulses;
- communicating with other systems or people;
- closely monitoring asynchronous external events.

Until now, microprocessor systems needed peripheral chips such as timer/counters, USARTs, or interrupt controllers to meet these needs. The 8051 integrates all of these capabilities on-chip!

#### Timer/Counters

There are two sixteen-bit multiple-mode Timer/Counters on the 8051, each consisting of a “High” byte (corresponding to the 8048 "T" register) and a low byte (similar to the 8048 prescaler, with the additional flexibility of being software-accessible). These registers are called, naturally enough, THO, TLO, TH1, and TL1. Each pair may be independently software programmed to any of a dozen modes with a mode register designated TMOD (Figure 7), and controlled with register TCON (Figure 8).

The timer modes can be used to measure time intervals, determine pulse widths, or initiate events, with one-microsecond resolution, up to a maximum interval of 65,536 instruction cycles (over 65 milliseconds). Longer delays may easily be accumulated through software. Configured as a counter, the same hardware will accumulate external events at frequencies from D.C. to 500 KHz, with up to sixteen bits of precision.

### Serial Port Interface

Each microcomputer contains a high-speed, full-duplex, serial port which is software programmable to function in four basic modes: shift-register I/O expander, 8-bit UART, 9-bit UART, or interprocessor communications link. The UART modes will interface with standard I/O devices (e.g. CRTs, teletypewriters, or modems) at data rates from 122 baud to 31 kilobaud. Replacing the standard 12 MHz crystal with a 10.7 MHz crystal allows 110 baud. Even or odd parity (if desired) can be included with simple bit-handling software routines. Inter-processor communications in distributed systems takes place at 187 kilobaud with hardware for automatic address/data message recognition. Simple TTL or CMOS shift registers provide low-cost I/O expansion at a super-fast 1 Mega-baud. The serial port operating modes are controlled by the contents of register SCON (Figure 9).

### Interrupt Capability and Control

(Interrupt capability is generally considered a CPU function. It is being introduced here since, from an applications point of view, interrupts relate more closely to peripheral and system interfacing.)
Gating control. When set, Timer/counter “x” is enabled only while “INTx” pin is high and “TRx” control bit is set. When cleared, timer/counter is enabled whenever “TRx” control bit is set.

Timer or Counter Selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from “Tx” input pin).

### Figure 7. TMOD—Timer/Counter Mode Register

<table>
<thead>
<tr>
<th>(MSB)</th>
<th>GATE</th>
<th>C/T</th>
<th>M1</th>
<th>M0</th>
<th>Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>(LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER 1</td>
<td>GATE</td>
<td>C/T</td>
<td>M1</td>
<td>M0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>MCS-48 Timer. “TLx” serves as five-bit prescaler.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>16-bit timer/counter. “THx” and “TLx” are cascaded; there is no prescaler.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8-bit auto-reload timer/counter. “THx” holds a value which is to be reloaded into “TLx” each time it overflows.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(Timer 0) TL0 is an eight-bit timer/counter controlled by the standard Timer 0 control bits. TH0 is an eight-bit timer only controlled by Timer 1 control bits.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>(Timer 1) Timer/counter 1 stopped.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF1</td>
<td>TCON.7</td>
<td>Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.</td>
</tr>
<tr>
<td>TR1</td>
<td>TCON.6</td>
<td>Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.</td>
</tr>
<tr>
<td>TF0</td>
<td>TCON.5</td>
<td>Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.</td>
</tr>
<tr>
<td>TR0</td>
<td>TCON.4</td>
<td>Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.</td>
</tr>
</tbody>
</table>

### Figure 8. TCON—Timer/Counter Control/Status Register

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE1</td>
<td>TCON.3</td>
<td>Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.</td>
</tr>
<tr>
<td>IT1</td>
<td>TCON.2</td>
<td>Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.</td>
</tr>
<tr>
<td>IE0</td>
<td>TCON.1</td>
<td>Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.</td>
</tr>
<tr>
<td>IT0</td>
<td>TCON.0</td>
<td>Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.</td>
</tr>
</tbody>
</table>
These peripheral functions allow special hardware to monitor real-time signal interfacing without bothering the CPU. For example, imagine serial data is arriving from one CRT while being transmitted to another, and one timer/counter is tallying high-speed input transitions while the other measures input pulse widths. During all of this the CPU is thinking about something else.

But how does the CPU know when a reception, transmission, count, or pulse is finished? The 8051 programmer can choose from three approaches.

TCON and SCON contain status bits set by the hardware when a timer overflows or a serial port operation is completed. The first technique reads the control register into the accumulator, tests the appropriate bit, and does a conditional branch based on the result. This "polling" scheme (typically a three-instruction sequence though additional instructions to save and restore the accumulator may sometimes be needed) will surely be familiar to programmers used to multi-chip microcomputer systems and peripheral controller chips. This process is rather cumbersome, especially when monitoring multiple peripherals.

As a second approach, the 8051 can perform a conditional branch based on the state of any control or status bit or input pin in a single instruction; a four instruction sequence could poll the four simultaneous happenings mentioned above in just eight microseconds.

Unfortunately, the CPU must still drop what it's doing to test these bits. A manager cannot do his own work well if he is continuously monitoring his subordinates; they should interrupt him (or her) only when they need attention or guidance. So it is with machines: ideally, the CPU would not have to worry about the peripherals until they require servicing. At that time, it would postpone the background task long enough to handle the appropriate device, then return to the point where it left off.

This is the basis of the third and generally optimal solution, hardware interrupts. The 8051 has five interrupt sources: one from the serial port when a transmission or reception is complete, two from the timers when overflows occur, and two from input pins INTO and INT1. Each source may be independently enabled or disabled to allow polling on some sources or at some times, and each may be classified as high or low priority. A high priority source can interrupt a low priority service routine; the manager's boss can interrupt conferences with subordinates. These options are selected by the interrupt enable and priority control registers, IE and IP (Figures 10 and 11).

Each source has a particular program memory address associated with it (Table 3), starting at 0003H (as in the 8048) and continuing at eight-byte intervals. When an event enabled for interrupts occurs the CPU automatically executes an internal subroutine call to the corresponding address. A user subroutine starting at this location (or jumped to from this location) then performs the instructions to service that particular source. After completing the interrupt service routine, execution returns to the background program.

<table>
<thead>
<tr>
<th>Interrupt Source</th>
<th>Service Routine Starting Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Reset)</td>
<td>0000H</td>
</tr>
<tr>
<td>External 0</td>
<td>0003H</td>
</tr>
<tr>
<td>Timer/Counter 0</td>
<td>000BH</td>
</tr>
<tr>
<td>External 1</td>
<td>0013H</td>
</tr>
<tr>
<td>Timer/Counter 1</td>
<td>001BH</td>
</tr>
<tr>
<td>Serial Port</td>
<td>0023H</td>
</tr>
</tbody>
</table>

Table 3. 8051 Interrupt Sources and Service Vectors
### IE—Interrupt Enable Register

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>IE.7</td>
<td>Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4-IE.0.</td>
</tr>
<tr>
<td>—</td>
<td>IE.6</td>
<td>(reserved)</td>
</tr>
<tr>
<td>—</td>
<td>IE.5</td>
<td>(reserved)</td>
</tr>
<tr>
<td>ES</td>
<td>IE.4</td>
<td>Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.</td>
</tr>
<tr>
<td>ET1</td>
<td>IE.3</td>
<td>Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.</td>
</tr>
</tbody>
</table>

### IP—Interrupt Priority Control Register

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Position</th>
<th>Name and Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>IP.7</td>
<td>(reserved)</td>
</tr>
<tr>
<td>—</td>
<td>IP.6</td>
<td>(reserved)</td>
</tr>
<tr>
<td>—</td>
<td>IP.5</td>
<td>(reserved)</td>
</tr>
<tr>
<td>PS</td>
<td>IP.4</td>
<td>Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.</td>
</tr>
<tr>
<td>PT1</td>
<td>IP.3</td>
<td>Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.</td>
</tr>
<tr>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PT0</td>
<td>IP.1</td>
<td>Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.</td>
</tr>
<tr>
<td>PX0</td>
<td>IP.0</td>
<td>External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INTO.</td>
</tr>
<tr>
<td>PX1</td>
<td>IP.2</td>
<td>External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.</td>
</tr>
</tbody>
</table>

Figure 10. IE—Interrupt Enable Register

Figure 11. IP—Interrupt Priority Control Register
3. INSTRUCTION SET AND ADDRESSING MODES

The 8051 instruction set is extremely regular, in the sense that most instructions can operate with variables from several different physical or logical address spaces. Before getting deeply enmeshed in the instruction set proper, it is important to understand the details of the most common data addressing modes. Whereas Table 4 summarizes the instruction sets broken down by functional group, this chapter starts with the addressing mode classes and builds to include the related instructions.

### Data Addressing Modes

MCS-51 assembly language instructions consist of an operation mnemonic and zero to three operands separated by commas. In two operand instructions the destination is specified first, then the source. Many byte-wide data
operations (such as ADD or MOV) inherently use the accumulator as a source operand and/or to receive the result. For the sake of clarity the letter "A" is specified in the source or destination field in all such instructions. For example, the instruction,

```
ADD A,<source>
```

will add the variable <source> to the accumulator, leaving the sum in the accumulator.

The operand designated "<source>" above may use any of four common logical addressing modes:

- Register—one of the working registers in the currently enabled bank.
- Direct—an internal RAM location, I/O port, or special-function register.
- Register-indirect—an internal RAM location, pointed to by a working register.
- Immediate data—an eight-bit constant incorporated into the instruction.

The first three modes provide access to the internal RAM and Hardware Register address spaces, and may therefore be used as source or destination operands; the last mode accesses program memory and may be a source operand only.

(It is hard to show a "typical application" of any instruction without involving instructions not yet described. The following descriptions use only the self-explanatory ADD and MOV instructions to demonstrate how the four addressing modes are specified and used. Subsequent examples will become increasingly complex.)

**Register Addressing**

The 8051 programmer has access to eight "working registers," numbered RO–R7. The least-significant three-bits of the instruction opcode indicate one register within this logical address space. Thus, a function code and operand address can be combined to form a short (one byte) instruction (Figure 12.a).

The 8051 assembly language indicates register addressing with the symbol Rn (where n is from 0 to 7) or with a symbolic name previously defined as a register by the EQUate or SET directives. (For more information on assembler directives see the Macro Assembler Reference Manual.)

**Example 1 — Adding Two Registers Together**

```
REGADR ADD CONTENTS OF REGISTER 1 TO CONTENTS OF REGISTER 0
REGADR MOV A,R0
ADD A,R1
MOV R0,A
```

There are four such banks of working registers, only one of which is active at a time. Physically, they occupy the first 32 bytes of on-chip data RAM (addresses 0–1FH). PSW bits 4 and 3 determine which bank is active. A hardware reset enables register bank 0; to select a different bank the programmer modifies PSW bits 4 and 3 accordingly.

**Example 2 — Selecting Alternate Memory Banks**

```
MOV PSW.00000000 SELECT BANK 2
```

Register addressing in the 8051 is the same as in the 8048 family, with two enhancements: there are four banks rather than one or two, and 16 instructions (rather than 12) can access them.

**Direct Byte Addressing**

Direct addressing can access any on-chip variable or hardware register. An additional byte appended to the opcode specifies the location to be used (Figure 12.b).

Depending on the highest order bit of the direct address byte, one of two physical memory spaces is selected. When the direct address is between 0 and 127 (00H–7FH) one of the 128 low-order on-chip RAM locations is used. (Future microcomputers based on the MCS-51™ architecture may incorporate more than 128 bytes of on-chip RAM. Even if this is the case, only the low-order 128 bytes will be directly addressable. The remainder would be accessed indirectly or via the stack pointer.)

**Example 3 — Adding RAM Location Contents**

```
DIRADR ADD CONTENTS OF RAM LOCATION 41H TO CONTENTS OF RAM LOCATION 40H
DIRADR MOV A,40H
ADD A,41H
MOV 40H,A
```

All I/O ports and special function, control, or status registers are assigned addresses between 128 and 255 (80H–0FFH). When the direct address byte is between these limits the corresponding hardware register is accessed. For example, Ports 0 and 1 are assigned direct addresses 80H and 90H, respectively. A complete list is presented in Table 5. Don't waste your time trying to memorize the addresses in Table 5. Since programs using absolute addresses for function registers would be difficult to write or understand, ASM51 allows and understands the abbreviations listed instead.

**Example 4 — Adding Input Port Data to Output Port Data**

```
PRTADR ADD DATA INPUT ON PORT 1 TO DATA PREVIOUSLY OUTPUT ON PORT 0
PRTADR MOV A,P0
ADD A,P1
MOV P0,A
```

Direct addressing allows all special-function registers in the 8051 to be read, written, or used as instruction operands. In general, this is the only method used for accessing I/O ports and special-function registers. If direct addressing is used with special-function register addresses other than those listed, the result of the instruction is undefined.
The 8048 does not have or need any generalized direct addressing mode, since there are only five special registers (BUS, P1, P2, PSW, & T) rather than twenty. Instead, 16 special 8048 opcodes control output bits or read or write each register to the accumulator. These functions are all subsumed by four of the 27 direct addressing instructions of the 8051.

Table 5. **8051 Hardware Register Direct Addresses**

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>80H*</td>
<td>Port 0</td>
</tr>
<tr>
<td>SP</td>
<td>81H</td>
<td>Stack Pointer</td>
</tr>
<tr>
<td>DPL</td>
<td>82H</td>
<td>Data Pointer (Low)</td>
</tr>
<tr>
<td>DPH</td>
<td>83H</td>
<td>Data Pointer (High)</td>
</tr>
<tr>
<td>TCON</td>
<td>88H</td>
<td>Timer register</td>
</tr>
<tr>
<td>TMOD</td>
<td>89H</td>
<td>Timer Mode register</td>
</tr>
<tr>
<td>TL0</td>
<td>8AH</td>
<td>Timer 0 Low byte</td>
</tr>
<tr>
<td>TL1</td>
<td>8BH</td>
<td>Timer 1 Low byte</td>
</tr>
<tr>
<td>TH0</td>
<td>8CH</td>
<td>Timer 0 High byte</td>
</tr>
<tr>
<td>TH1</td>
<td>8DH</td>
<td>Timer 1 High byte</td>
</tr>
<tr>
<td>PI</td>
<td>90H*</td>
<td>Port 1</td>
</tr>
<tr>
<td>SCON</td>
<td>98H*</td>
<td>Serial Port Control register</td>
</tr>
<tr>
<td>SBUF</td>
<td>99H</td>
<td>Serial Port data Buffer</td>
</tr>
<tr>
<td>P2</td>
<td>0A0H*</td>
<td>Port 2</td>
</tr>
<tr>
<td>IE</td>
<td>0A8H*</td>
<td>Interrupt Enable register</td>
</tr>
<tr>
<td>P3</td>
<td>0B0H*</td>
<td>Port 3</td>
</tr>
<tr>
<td>IP</td>
<td>0B8H*</td>
<td>Interrupt Priority register</td>
</tr>
<tr>
<td>PSW</td>
<td>0D0H*</td>
<td>Program Status Word</td>
</tr>
<tr>
<td>ACC</td>
<td>0E0H*</td>
<td>Accumulator (direct address)</td>
</tr>
<tr>
<td>B</td>
<td>0F0H*</td>
<td>B register</td>
</tr>
</tbody>
</table>

* = bit addressable register.

### Register-Indirect Addressing

How can you handle variables whose locations in RAM are determined, computed, or modified while the program is running? This situation arises when manipulating sequential memory locations, indexed entries within tables in RAM, and multiple precision or string operations. Register or Direct addressing cannot be used, since their operand addresses are fixed at assembly time.

The 8051 solution is “register-indirect RAM addressing.” R0 and R1 of each register bank may operate as index or pointer registers, their contents indicating an address into RAM. The internal RAM location so addressed is the actual operand used. The least significant bit of the instruction opcode determines which register is used as the “pointer” (Figure 12.c).

In the 8051 assembly language, register-indirect addressing is represented by a commercial “at” sign (“@”) preceding R0, R1, or a symbol defined by the user to be equal to R0 or R1.

**Example 5 — Indirect Addressing**

```assembly
INADR ADD CONTENTS OF MEMORY LOCATION
    ADDRESSED BY REGISTER 1 TO REGISTERS Location and
    ADDRESSED BY REGISTER 0

INADR MOV A, @R1
ADD A, R1
MOV B, @R1
```

Indirect addressing on the 8051 is the same as in the 8048 family, except that all eight bits of the pointer register contents are significant; if the contents point to a nonexistent memory location (i.e., an address greater than 7FH on the 8051) the result of the instruction is undefined. (Future microcomputers based on the MCS-51™ architecture could implement additional memory in the on-chip RAM logical address space at locations above 7FH.) The 8051 uses register-indirect addressing for five new instructions plus the 13 on the 8048.

### Immediate Addressing

When a source operand is a constant rather than a variable (i.e.—the instruction uses a value known at assembly time), then the constant can be incorporated into the instruction. An additional instruction byte specifies the value used (Figure 12.d).

The value used is fixed at the time of ROM manufacture or EPROM programming and may not be altered during program execution. In the assembly language immediate operands are preceded by a number sign (“#”). The operand may be either a numeric string, a symbolic variable, or an arithmetic expression using constants.

**Example 6 — Adding Constants Using Immediate Addressing**

```assembly
IMMAOR ADD THE CONSTANT 0D (DECIMAL) TO THE CONSTANT 34 (DECIMAL) AND LEAVE SUM IN ACCUMULATOR
IMMAOR MOV A, #12
ADD A, #34
```

The preceding example was included for consistency; it has little practical value. Instead, ASM51 could compute the sum of two constants at assembly time.

**Example 7 — Adding Constants Using ASM51 Capabilities**

```assembly
ASMSUM LOAD ACC WITH THE SUM OF THE CONSTANT 12 (DECIMAL) AND THE CONSTANT 34 (DECIMAL)
ASMSUM MOV A, #12+34
```

Figure 12. Data Addressing Machine Code Formats

AFN-01502A-19

10-17
Addressing Mode Combinations

The above examples all demonstrated the use of the four data-addressing modes in two-operand instructions (MOV, ADD) which use the accumulator as one operand. The operations ADDC, SUBB, ANL, ORL, and XRL (all to be discussed later) could be substituted for ADD in each example. The first three modes may be also be used for the XCH operation or, in combination with the Immediate Addressing mode (and an additional byte), loaded with a constant. The one-operand instructions INC and DEC, DJNZ, and CJNE may all operate on the accumulator, or may specify the Register, Direct, and Register-indirect addressing modes. Exception: as in the 8048, DJNZ cannot use the accumulator or indirect addressing. (The PUSH and POP operations cannot inherently address the accumulator as a special register either. However, all three can directly address the accumulator as one of the twenty special-function registers by putting the symbol "ACC" in the operand field.)

Advantages of Symbolic Addressing

Like most assembly or higher-level programming languages, ASM51 allows instructions or variables to be given appropriate, user-defined symbolic names. This is done for instruction lines by putting a label followed by a colon (":") before the instruction proper, as in the above examples. Such symbols must start with an alphabetic character (remember what distinguished BACH from OBACH?), and may include any combination of letters, numbers, question marks ("?"), and underscores ("_"). For very long names only the first 31 characters are relevant.

Assembly language programs may intermix upper- and lower-case letters arbitrarily, but ASM51 converts both to upper-case. For example, ASM51 will internally process an "I" for an "i" and, of course, "A_TOOTH" for "a_tooth."

The underscore character makes symbols easier to read and can eliminate potential ambiguity (as in the label for a subroutine to switch two entries on a stack, "S_EXCHANGE"). The underscore is significant, and would distinguishes between otherwise-identical character strings.

ASM51 allows all variables (registers, ports, internal or external RAM addresses, constants, etc.) to be assigned labels according to these rules with the EQUate or SET directives.

Example 8 — Symbolic Addressing of Variables Defined as RAM Locations

<table>
<thead>
<tr>
<th>VAR_O</th>
<th>SET</th>
<th>20H</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR_1</td>
<td>SET</td>
<td>21H</td>
</tr>
<tr>
<td>SYMB_1 ADD CONTENTS OF VAR_1 TO CONTENTS OF VAR_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYMB_1 MOV A: VAR_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD A: VAR_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV VAR_0: A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notice from Table 4 that the MCS-51™ instruction set has relatively few instruction mnemonics (abbreviations) for the programmer to memorize. Different data types or addressing modes are determined by the operands specified, rather than variations on the mnemonic. For example, the mnemonic "MOV" is used by 18 different instructions to operate on three data types (bit, byte, and address). The fifteen versions which move byte variables between the logical address spaces are diagrammed in Figure 13. Each arrow shows the direction of transfer from source to destination.

Notice also that for most instructions allowing register addressing there is a corresponding direct addressing instruction and vice versa. This lets the programmer begin writing 8051 programs as if (s)he has access to 128 different registers. When the program has evolved to the point where the programmer has a fairly accurate idea how often each variable is used, (s)he may allocate the working registers in each bank to the most "popular" variables. (The assembly cross-reference option will show exactly how often and where each symbol is referenced.)

If symbolic addressing is used in writing the source program only the lines containing the symbol definition will need to be changed; the assembler will produce the appropriate instructions even though the rest of the program is left untouched. Editing only the first two lines of Example 8 will shrink the six-byte code segment produced in half.

How are instruction sets "counted"? There is no standard practice; different people assessing the same CPU using different conventions may arrive at different totals.

Each operation is then broken down according to the different addressing modes (or combinations of addressing modes) it can accommodate. The "CLR" mnemonic is used by two instructions with respect to bit variables ("CLR C" and "CLR bit") and once ("CLR A") with regards to bytes. This expansion yields the 111 separate instructions of Table 4.

The method used for the MCS-51* instruction set first breaks it down into "operations": a basic function applied to a single data type. For example, the four versions of the ADD instruction are grouped to form one operation — addition of eight-bit variables. The six forms of the ANL instruction for byte variables make up a different operation; the two forms of ANL which operate on bits are considered still another. The MOV mnemonic is used by three different operation classes, depending on whether bit, byte, or 16-bit values are affected. Using this terminology the 8051 can perform 51 different operations.
Arithmetic Instruction Usage — ADD, ADDC, SUBB and DA

The ADD instruction adds a byte variable with the accumulator, leaving the result in the accumulator. The carry flag is set if there is an overflow from bit 7 and cleared otherwise. The AC flag is set to the carry-out from bit 3 for use by the DA instruction described later. ADDC adds the previous contents of the carry flag with the two byte variables, but otherwise is the same as ADD.

The SUBB (subtract with borrow) instruction subtracts the byte variable indicated and the contents of the carry flag together from the accumulator, and puts the result back in the accumulator. The carry flag serves as a “Borrow Required” flag during subtraction operations; when a greater value is subtracted from a lesser value (as in subtracting 5 from 1) requiring a borrow into the highest order bit, the carry flag is set; otherwise it is cleared.

When performing signed binary arithmetic, certain combinations of input variables can produce results which seem to violate the Laws of Mathematics. For example, adding 7FH (127) to itself produces a sum of 0FEH, which is the two's complement representation of -2 (refer back to Table 2)! In “normal” arithmetic, two positive values can't have a negative sum. Similarly, it is normally impossible to subtract a positive value from a negative value and leave a positive result — but in two's complement there are instances where this too may happen. Fundamentally, such anomalies occur when the magnitude of the result’s value is too great to “fit” into the seven bits allowed for it; there is no one-byte two's complement representation for 254, the true sum of 127 and 127.

The MCS-51™ processors detect whether these situations occur and indicate such errors with the OV flag. (OV may be tested with the conditional jump instructions JB and JNB, described under the Boolean Processor chapter.)

At a hardware level, OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6. When adding signed integers this indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands; on SUBB this indicates a negative result after subtracting a negative number from a positive number, or a positive result when a positive number is subtracted from a negative number.

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple precision calculations by repeating the operation with successively higher-order operand bytes. In either case, the carry must be cleared before the first iteration.

If the input data for a multiple precision operation is an unsigned string of positive numbers, upon completion the carry flag will be set if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data (i.e., if the most significant bit of the original input data indicates the sign of the string), the overflow flag will be set if overflow or underflow occurred.

Example 10 — String Subtraction with Signed Overflow Detection

Decimal addition is possible by using the DA instruction in conjunction with ADD and/or ADDC. The eight-bit binary value in the accumulator resulting from an earlier addition of two variables (each a packed BCD digit-pair) is adjusted to form two BCD digits of four bits each. If the contents of accumulator bits 3-0 are greater than nine (10000100xxxx1111), or if the AC flag had been set, six is added to the accumulator producing the proper BCD digit in the low-order nibble. (This addition might itself set — but would not clear — the carry flag.) If the carry flag is set, or if the four high-order bits now exceed nine (10000100xxxx1111xxxx), these bits are incremented by six. The carry flag is left set if originally set or if either addition of six produces a carry out of the highest-order bit, indicating the sum of the original two BCD variables is greater than or equal to decimal 100.
Example 11 — Two Byte Decimal Add with Registers and Constants

Example 12 — Use of DIV Instruction for Radix Conversion

**Multiplication and Division**

The instruction "MUL AB" multiplies the unsigned eight-bit integer values held in the accumulator and B-registers. The low-order byte of the sixteen-bit product is left in the accumulator, the higher-order byte in B. If the high-order eight-bits of the product are all zero the overflow flag is cleared; otherwise it is set. The programmer can poll OV to determine when the B register is non-zero and must be processed.

"DIV AB" divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in the B-register. The integer part of the quotient is returned in the accumulator; the remainder in the B-register. If the B-register originally contained 00H then the overflow flag will be set to indicate a division error, and the values returned will be undefined. Otherwise OV is cleared.

The divide instruction is also useful for purposes such as radix conversion or separating bit fields of the accumulator. A short subroutine can convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one register (HUNDREDS) and the ten's and one's digits returned as packed BCD in another (TENONE).

Example 13 — Implementing a BCD Multiply Using MPY and DIV

**Logical Byte Operations — ANL, ORL, XRL**

The instructions ANL, ORL, and XRL perform the logical functions AND, OR, and/or Exclusive-OR on the two byte variables indicated, leaving the results in the first. No flags are affected. (A word to the wise — do not vocalize the first two mnemonics in mixed company.)

These operations may use all the same addressing modes as the arithmetics (ADD, etc.) but unlike the arithmetics, they are not restricted to operating on the accumulator. Directly addressed bytes may be used as the destination with either the accumulator or a constant as the source. These instructions are useful for clearing (ANL), setting (ORL), or complementing (XRL) one or more bits in a RAM, output ports, or control registers. The pattern of bits to be affected is indicated by a suitable mask byte. Use immediate addressing when the pattern to be affected is known at assembly time (Figure 14); use the accumulator versions when the pattern is computed at run-time.

I/O ports are often used for parallel data in formats other than simple eight-bit bytes. For example, the low-order five bits of port 1 may output an alphabetic character code (hopefully) without disturbing bits 7-5. This can be a simple two-step process. First, clear the low-order five pins with an ANL instruction; then set those pins corresponding to ones in the accumulator. (This example assumes the three high-order bits of the accumulator are originally zero.)

Example 14 — Reconfiguring Port Size with Logical Byte Instructions

![Figure 14. Instruction Pattern for Logical Operation Special Addressing Modes](AFN-01502A-22)
In this example, low-order bits remaining high may "glitch" low for one machine cycle. If this is undesirable, use a slightly different approach. First, set all pins corresponding to accumulator one bits, then clear the pins corresponding to zeroes in low-order accumulator bits. Not all bits will change from original to final state at the same instant, but no bit makes an intermediate transition.

Example 15 — Reconfiguring I/O Port Size without Glitching

Program Control — Jumps, Calls, Returns

Whereas the 8048 only has a single form of the simple jump instruction, the 8051 has three. Each causes the program to unconditionally jump to some other address. They differ in how the machine code represents the destination address.

LJMP (Long Jump) encodes a sixteen-bit address in the second and third instruction bytes (Figure 15.a); the destination may anywhere in the 64 Kilobyte program memory address space.

The two-byte AJMP (Absolute Jump) instruction encodes its destination using the same format as the 8048: address bits 10 through 8 form a three bit field in the opcode and address bits 7 through 0 form the second byte (Figure 15.b). Address bits 15-12 are unchanged from the (incremented) contents of the P.C., so AJMP can only be used when the destination is known to be within the same 2K memory block. (Otherwise ASM51 will point out the error.)

A different two-byte jump instruction is legal with any nearby destination, regardless of memory block boundaries or "pages." SJMP (Short Jump) encodes the destination with a program counter-relative address in the second byte (Figure 15.c). The CPU calculates the destination at run-time by adding the signed eight-bit displacement value to the incremented P.C. Negative offset values will cause jumps up to 128 bytes backwards; positive values up to 127 bytes forwards. (SJMP with 00H in the machine code offset byte will proceed with the following instruction).

In keeping with the 8051 assembly language goal of minimizing the number of instruction mnemonics, there is a "generic" form of the three jump instructions. ASM51 recognizes the mnemonic LJMP as a "pseudo-instruction," translating it into the machine instructions LJMP, AJMP, or SJMP, depending on the destination address.

Like SJMP, all conditional jump instructions use relative addressing. JZ (Jump if Zero) and JNZ (Jump if Not Zero) monitor the state of the accumulator as implied by their names, while JC (Jump on Carry) and JNC (Jump on No Carry) test whether or not the carry flag is set. All four are two-byte instructions, with the same format as Figure 15.c. JB (Jump on Bit), JNB (Jump on No Bit) and JBC (Jump on Bit then Clear Bit) can test any status bit or input pin with a three byte instruction; the second byte specifies which bit to test and the third gives the relative offset value.

There are two subroutine-call instructions, LCALL (Long Call) and ACALL (Absolute Call). Each increments the P.C. to the first byte of the following instruction, then pushes it onto the stack (low byte first). Saving both bytes increments the stack pointer by two. The subroutine's starting address is encoded in the same ways as LJMP and AJMP. The generic form of the call operation is the mnemonic CALL, which ASM51 will translate into LCALL or ACALL as appropriate.

The return instruction RET pops the high- and low-order bytes of the program counter successively from the stack, decrementing the stack pointer by two. Program execution continues at the address previously pushed; the first byte of the instruction immediately following the call.

When an interrupt request is recognized by the 8051 hardware, two things happen. Program control is automatically "vectorized" to one of the interrupt service routine starting addresses by, in effect, forcing the CPU to process a LCALL instead of the next instruction. This automatically stores the return address on the stack. (Unlike the 8048, no status information is automatically saved.)

Secondly, the interrupt logic is disabled from accepting any other interrupts from the same or lower priority. After completing the interrupt service routine, executing an RETI (Return from Interrupt) instruction will return execution to the point where the background program was interrupted — just like RET — while restoring the interrupt logic to its previous state.
Operate-and-branch instructions — CJNE, DJNZ

Two groups of instructions combine a byte operation with a conditional jump based on the results.

CJNE (Compare and Jump if Not Equal) compares two byte operands and executes a jump if they disagree. The carry flag is set following the rules for subtraction: if the unsigned integer value of the first operand is less than that of the second it is set; otherwise, it is cleared. However, neither operand is modified.

The CJNE instruction provides, in effect, a "special case" statement. This instruction may be executed repeatedly, comparing the code variable to a list of "special case" value: the code segment following the instruction (up to the destination label) will be executed only if the operands match. Comparing the accumulator or a register to a series of constants is a convenient way to check (for special handling or error conditions; if none of the cases match the program will continue with "normal" processing.

A typical example might be a word processing device which receives ASCII characters through the serial port and drives a thermal hard-copy printer. A standard routine translates "printing" characters to bit patterns, but control characters (<DEL>, <CR>, <LF>, <BEL> or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters.

Example 16 — Case Statements Using CJNE

<table>
<thead>
<tr>
<th>CHAR</th>
<th>CJNE</th>
<th>CHAR. A7FH, INTP_1</th>
<th>SPECIAL ROUTINE FOR RUBOUT CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. R07H, INTP_2</td>
<td>SPECIAL ROUTINE FOR RUBOUT CODE</td>
</tr>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. AOH, INTP_3</td>
<td>SPECIAL ROUTINE FOR LFDCC CODE</td>
</tr>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. MO7H, INTP_4</td>
<td>SPECIAL ROUTINE FOR LFCC CODE</td>
</tr>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. 81H, INTP_5</td>
<td>SPECIAL ROUTINE FOR ESCAPE CODE</td>
</tr>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. 82H, INTP_6</td>
<td>SPECIAL ROUTINE FOR SPACE CODE</td>
</tr>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. MO7H, INTP_4</td>
<td>SPECIAL ROUTINE FOR LFCC CODE</td>
</tr>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. AOH, INTP_4</td>
<td>SPECIAL ROUTINE FOR LFCC CODE</td>
</tr>
<tr>
<td>RET</td>
<td>CJNE</td>
<td>CHAR. R07H, INTP_2</td>
<td>SPECIAL ROUTINE FOR RUBOUT CODE</td>
</tr>
</tbody>
</table>

DJNZ (Decrement and Jump if Not Zero) decrements the register or direct address indicated and jumps if the result is not zero, without affecting any flags. This provides a simple means for executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. For example, a 99-usec. software delay loop can be added to code forcing an I/O pin low with only two instructions.

Example 17 — Inserting a Software Delay with DJNZ

<table>
<thead>
<tr>
<th>CLR</th>
<th>MOV</th>
<th>R2, #49</th>
</tr>
</thead>
<tbody>
<tr>
<td>DJNZ</td>
<td>RET</td>
<td>R2, #6</td>
</tr>
</tbody>
</table>

The dollar sign in this example is a special character meaning "the address of this instruction." It is useful in eliminating instruction labels on the same or adjacent source lines. CJNE and DJNZ (like all conditional jumps) use program-counter relative addressing for the destination address.

Stack Operations — PUSH, POP

The PUSH instruction increments the stack pointer by one, then transfers the contents of the single byte variable indicated (direct addressing only) into the internal RAM location addressed by the stack pointer. Conversely, POP copies the contents of the internal RAM location addressed by the stack pointer to the byte variable indicated, then decrements the stack pointer by one.

(Stack Addressing follows the same rules, and addresses the same locations as Register-indirect. Future microcomputers based on the MCS-51™ CPU could have up to 256 bytes of RAM for the stack.)

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save register states on the stack.

Example 18 — Use of the Stack for Status Saving on Interrupts

LOC_INT EQU $         ; REMEMBER LOCATION COUNTER
LOC_INT RET             | STARTING ADDRESS FOR INTERRUPT ROUTINE
ORG 0000H             | JUMP TO ACTUAL SERVICE ROUTINE LOCATED
LMP                | ELSEWHERE
LOC_INT  MOV             | RESTORE LOCATION COUNTER
ORG LOC_INT           | RESTORE STATUS REGISTER TO ORIGINAL NOTATION
push                | SAVE PSW (NOTE: STACK REGISTER)
PUSH                | B — SAVE B REGISTER
PUSH                | DPL — SAVE DATA POINTER
PUSH                | DPH — SAVE ACC!
mov                | PSM. 00001000B — SELECT REGISTER BANK!
PUSH                | RESTORE REGISTERS IN REVERSE ORDER
PUSH                | DPL
PUSH                | ACC
PUSH                | PSW — RESTORE PSW AND RE-SELECT ORIGINAL
RETI                | SETTLE BANK
RETURN               | RETURN TO MAIN PROGRAM AND RESTORE
INTERCEPT            | INTERRUPT LOCID

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 16; SP would contain 26H.

The example shows the most general situation, if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers would not be necessary.

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer.

AFN-01502A-24
Figure 16. Stack contents during interrupt

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the following subroutine reads out a parameter stored on the stack by the calling program, uses the low order bits to access a local look-up table holding bit patterns for driving the coils of a four phase stepper motor, and stores the appropriate bit pattern back in the same position on the stack before returning. The accumulator contents are left unchanged.

Example 19 — Passing Variable Parameters to Subroutines Using the Stack

If the position of the motor is determined by the contents of variable POSM1 (a byte in internal RAM) and the position of a second motor on Port 2 is determined by the data input to the low-order nibble of Port 2, a six-instruction sequence could update them both.

Example 21 — Loading and Unloading Stack Direct from I/O Ports

The data pointer can be loaded with a 16-bit value using the instruction MOV DPTR, #data16. The data is stored in the second and third instruction bytes, high-order byte first. The data pointer is incremented by INC DPTR. A 16-bit increment is performed; an overflow from the low byte will carry into the high-order byte. Neither instruction affects any flags.

The MOVC (Move Constant) instructions (MOVC A,@A+DPTR and MOVC A,@A+PC) read into the accumulator bytes of data from the program memory logical address space. Both use a form of indexed addressing: the former adds the unsigned eight-bit accumulator contents with the sixteen-bit data pointer register, and uses the resulting sum as the address from which the byte is fetched. A sixteen-bit addition is performed; a carry-out from the low-order eight bits may propagate through higher-order bits, but the contents of the DPTR are not altered. The latter form uses the incremented program counter as the "base" value instead of the DPTR (figure 17). Again, neither version affects the flags.

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result after. A motor on Port 1 may be initialized by placing the desired position (zero) on the stack before calling the subroutine and outputing the results directly to a port afterwards.

Example 20 — Sending and Receiving Data Parameters Via the Stack
Each can be part of a three step sequence to access look-up tables in ROM. To use the DPTR-relative version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute MOVCA, @A+DPTR. Unlike the similar MOVPA instructions in the 8048, the table may be located anywhere in program memory. The data pointer may be loaded with a constant for short tables. Or to allow more complicated data structures, or tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-relative version has the advantage of not affecting the data pointer. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction to separating them to the accumulator; then execute the instruction. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction to separating them to the accumulator; then execute the MOVC, @A+PC instruction.

Let’s look at a non-trivial situation where this instruction would be used. Some applications store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in a linear (one-dimensional) vector in program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry’s memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula:

Entry Address = BASE + (NDIMEN x INDEXI) + INDEXJ

The code shown below can access any array with less than 255 entries (i.e., an 11x21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

Example 22 — Use of MPY and Data Pointer Instructions to Access Entries from a Multi-Dimensional Look-Up Table in ROM

There are several different means for branching to sections of code determined or selected at run time. The single destination addresses incorporated into conditional and unconditional jumps are, of course, determined at assembly time. Each has advantages for different applications.

The most common is an N-way conditional jump based on some variable, with all of the potential destinations known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the MOVC and an indirect jump instruction, using a short table of one byte offset values in ROM to indicate the relative starting addresses of the several routines.

JMP, @A+DPTR is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like MOVC, @A+DPTR. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed; a carry out from the low-order eight bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types and sizes of buffer memory for different speeds and options.

Example 23 — N-Way Branch and Computed Jump Instructions via JMP, @ADPTR

Note that this approach is suitable whenever the size of jump table plus the length of the alternate routines is less than 256 bytes. The jump table and routines may be located anywhere in program memory, independent of 256-byte program memory pages.
For applications where up to 128 destinations must be selected, all of which reside in the same 2K page of program memory which may be reached by the two-byte absolute jump instructions, the following technique may be used. In the above mentioned printing terminal example, this sequence could "parse" 128 different codes for ASCII characters arriving via the 8051 serial port.

Example 24 — N-Way Branch with 128 Optional Destinations

```
OPTION EQU #3

JMP:128: MOV A,OPTION
    INC DPTR
    INCB #1
    JMP PROC02

PROC02:
    MOV A,128
    JBC 128

PROC00:
    MOV A,0
    JBC 0

PROC01:
    MOV A,1
    JBC 1

PROC07:
    MOV A,7
    JBC 7

PROC7F:
    MOV A,7F
    JBC 7F
```

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

In those rare situations where even 128 options are insufficient, or where the destination routines may cross a 2K page boundary, the above approach may be modified slightly as shown below.

Example 25 — 256-Way Branch Using Address Look-Up Tables

```
RTMP EQU #7

JMP:256: MOV DPTR,#ADRTBL
    MOV A,RTMP
    JNC #4
    JNC #10
    JNC #20
    JNC #40
    JNC #80
    JNC #180
    JNC #280

ADRTBL:
    DW PROC00
    DW PROC01
    DW PROC02
    DW PROC03
    DW PROC04
    DW PROC05
    DW PROC06
    DW PROC07
    DW PROC08
    DW PROC09
    DW PROC0A
    DW PROC0B
    DW PROC0C
    DW PROC0D
    DW PROC0E
    DW PROC0F
```

Prior to the introduction of the MCS-51™ family, nice number-crunchers made bad bit-bangers and vice versa. The 8051 is the industry’s first single-chip microcomputer designed to crunch and bang. (In some circles, the latter technique is also referred to as “bit-twiddling”. Either is correct.)

**Direct Bit Addressing**

A number of instructions operate on Boolean (one-bit) variables, using a direct bit addressing mode comparable to direct byte addressing. An additional byte appended to the opcode specifies the Boolean variable, I/O pin, or control bit used. The state of any of these bits may be tested for “true” or “false” with the conditional branch instructions JB (Jump on Bit) and JNB (Jump on Not Bit). The JBC (Jump on Bit, and Clear) instruction combines a test-for-true with an unconditional clear.

As in direct byte addressing, bit 7 of the address byte switches between two physical address spaces. Values between 0 and 127 (00H-7FH) define bits in internal RAM locations 20H to 2FH (Figure 18a); address bytes between 128 and 255 (80H-0FFH) define bits in the 2 x “special-function” register address space (Figure 18b). If no 2 x “special-function” register corresponds to the direct bit address used the result of the instruction is undefined.

Bits so addressed have many wondrous properties. They may be set, cleared, or complemented with the two byte instructions SETB, CLR, or CPL. Bits may be moved to and from the carry flag with MOV. The logical ANL and ORL functions may be performed between the carry and either the addressed bit or its complement.

**Bit Manipulation Instructions — MOV**

The “MOV” mnemonic can be used to load an addressable bit into the carry flag (“MOV C, bit”) or to copy the state of the carry to such a bit (“MOV bit, C”). These instructions are often used for implementing serial I/O algorithms via software or to adapt the standard I/O port structure.

It is sometimes desirable to “re-arrange” the order of I/O pins because of considerations in laying out printed circuit boards. When interfacing the 8051 to an immediately adjacent device with “weighted” input pins, such as keyboard column decoder, the corresponding pins are likely to be not aligned (Figure 19).

There is a trade-off in “scrambling” the interconnections with either interwoven circuit board traces or through software. This is extremely cumbersome (if not impossible) to do with byte-oriented computer architectures. The 8051’s unique set of Boolean instructions makes it simple to move individual bits between arbitrary locations.

**4. BOOLEAN PROCESSING INSTRUCTIONS**

The commonly accepted terms for tasks at either end of the computational vs. control application spectrum are, respectively, “number-crunching” and “bit-banging.”
Virtually all hardware designers are familiar with the problem of solving complex functions using combinatorial logic. The technologies involved may vary greatly, from multiple contact relay logic, vacuum tubes, TTL, or CMOS to more esoteric approaches like fluidics, but in each case the goal is the same: a Boolean (true/false) function is computed on a number of Boolean variables.
Figure 20. Implementations of Boolean functions

Figure 20 shows the logic diagram for an arbitrary function of six variables named U through Z using standard logic and relay logic symbols. Each is a solution of the equation,

\[ Q = (U \cdot (V + W)) + (X \cdot \overline{Y}) + \overline{Z} \]

(While this equation could be reduced using Karnaugh Maps or algebraic techniques, that is not the purpose of this example. Even a minor change to the function equation would require re-reducing from scratch.)

Most digital computers can solve equations of this type with standard word-wide logical instructions and conditional jumps. Still, such software solutions seem somewhat sloppy because of the many paths through the program the computation can take.

Assume U and V are input pins being read by different input ports, W and X are status bits for two peripheral controllers (read as I/O ports), and Y and Z are software flags set or cleared earlier in the program. The end result must be written to an output pin on some third port.

For the sake of comparison we will implement this function with software drawn from three proper subsets of the MCS-51™ instruction set. The first two implementations follow the flow chart shown in Figure 21. Program flow would embark on a route down a test-and-branch tree and leaves either the “True” or “Not True” exit ASAP. These exits then write the output port with the data previously written to the same port with the result bit respectively one or zero.

In the first case, we assume there are no instructions for addressing individual bits other than special flags like the carry. This is typical of many older microprocessors and mainframe computers designed for number-crunching. MCS-51™ mnemonics are used here, though for most other machines the issue would be even further clouded by their use of operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the universal MOV.

Figure 21. Flow chart for tree-branching logic implementation
Cumbersome, to say the least, and error prone. It would be hard to prove the above example worked in all cases without an exhaustive test.

Each move/mask/conditional jump instruction sequence may be replaced by a single bit-test instruction thanks to direct bit addressing. But the algorithm would be equally convoluted.

Example 28 — Software Solution to Logic Function of Figure 20, Using only Bit-Test Instructions

A more elegant and efficient 8051 implementation uses the Boolean ANL and ORL functions to generate the output function using straight-line code. These instructions perform the corresponding logical operations between the carry flag (“Boolean Accumulator”) and the addressed bit, leaving the result in the carry. Alternate forms of each instruction (specified in the assembly language by placing a slash before the bit name) use the complement of the bit’s state as the input operand.

These instructions may be “strung together” to simulate a multiple input logic gate. When finished, the carry flag contains the result, which may be moved directly to the destination or output pin. No flow chart is needed — it is simple to code directly from the logic diagrams in Figure 20.

Example 29 — Software Solution to Logic Function of Figure 20, Using the MCS-51 (TM) Unique Logical Instructions on Boolean Variables

Simplicity itself. Fast, flexible, reliable, easy to design, and easy to debug.

The Boolean features are useful and unique enough to warrant a complete Application Note of their own. Additional uses and ideas are presented in Application Note AP-70, Using the Intel® MCS-51® Boolean Processing Capabilities, publication number 121519.

5. ON-CHIP PERIPHERAL FUNCTION OPERATION AND INTERFACING

I/O Ports

The I/O port versatility results from the “quasi-bidirectional” output structure depicted in Figure 22. (This is effectively the structure of ports 1, 2, and 3 for normal I/O operations. On port 0 resistor R2 is disabled except during multiplexed bus operations, providing

![Figure 22. Pseudo-bidirectional I/O port circuitry](AFN-01502A-30)
essentially open-collector outputs. For full electrical characteristics see the User's Manual.)

An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open-collector output with a pull-up resistor to Vcc.)

R2 and Q2 represent an "active pull-up" device enabled momentarily when a 0 previously output changes to a 1. This "jerks" the output pin to a 1 level more quickly than the passive pull-up, improving rise-time significantly if the pin is driving a capacitive load. Note that the active pull-up is only activated on 0-to-1 transitions at the output latch (unlike the 8048, in which Q2 is activated whenever a 1 is written out).

Operations using an input port or pin as the source operand use the logic level of the pin itself, rather than the output latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedance, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously output. To use a pin for input, the corresponding output latch must be set. The external device may then drive the pin with either a high or low logic signal. Thus the same port may be used as both input and output by writing ones to all pins used as inputs on output operations, and ignoring all pins used as output on an input operation.

In one operand instructions (INC, DEC, DJNZ and the Boolean CPL) the output latch rather than the input pin level is used as the source data. Similarly, two operand instructions using the port as both one source and the destination (ANL, ORL, XRL) use the output latches. This ensures that latch bits corresponding to pins used as inputs will not be cleared in the process of executing these instructions.

The Boolean operation JBC tests the output latch bit, rather than the input pin, in deciding whether or not to jump. Like the byte-wise logical operations, Boolean operations which modify individual pins of a port leave the other bits of the output latch unchanged.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected (Figure 23) and the protocol may be emulated with simple software.

**Example 30 — Mixing Parallel Output, Input, and Control Strobes on Port 2**

**Serial Port and Timer applications**

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

This is best illustrated through an arbitrary example. Assume the 8051 will communicate with a CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. This results in a character rate of 2400/10=240 characters per second.

For the sake of clarity, the transmit and receive subroutines are driven by simple-minded software status polling code rather than interrupts. (It might help to refer back to Figures 7-9 showing the control word formats.) The serial port must be initialized to 8-bit UART mode (M0, M1=01), enabled to receive all messages (M2=0, REN=1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. This can all be set up with one instruction.

**Example 31 — Serial Port Mode and Control Bits**

For 8-BIT UART MODE

& SET TRANSMIT READY FLAG

SPIRINT MOV A SCON.001010100B

---

**Figure 23. Connecting an 8051 with an 8243 I/O Expander**

10-29
Timer 1 will be used in auto-reload mode as a data rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1 MHz internal clock by 32 x (desired data rate):

\[
\frac{1 \times 10^6}{(32)(2400)}
\]

which equals 13.02 rounded down to 13 instruction cycles. The timer must reload the value -13, or 0F3H. (ASM51 will accept both the signed decimal or hexadecimal representations.)

Example 32 — Initializing Timer Mode and Control Bits

```
.TINIT INITIALIZE TIMER 1 FOR AUTO-RELOAD AT 32x2400 Hz
.MOV TCON, #11010010B
.MOV TH0, #-13
.SETB TR0
```

A simple subroutine to transmit the character passed to it in the accumulator must first compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and return. This is nearly as easy said as done.

Example 33 — Code for UART Output, Adding Parity, Transmitter Loading

```
.SP_OUT ADD ODD PARITY TO ACC AND TRANSMIT WHEN SERIAL PORT READY
.SP_IN INPUT NEXT CHARACTER FROM SERIAL PORT SET CARRY IF ODD-PARITY ERROR
```

A simple minded routine to wait until a character is received, set the carry flag if there is an odd-parity error, and return the masked seven-bit code in the accumulator is equally short.

Example 34 — Code for UART Reception and Parity Verification

```
```

6. SUMMARY

This Application Note has described the architecture, instruction set, and on-chip peripheral features of the first three members of the MCS-51™ microcomputer family. The examples used throughout were admittedly (and necessarily) very simple. Additional examples and techniques may be found in the MCS-51™ User’s Manual and other application notes written for the MCS-48™ and MCS-51™ families.

Since its introduction in 1977, the MCS-48™ family has become the industry standard single-chip microcomputer. The MCS-51™ architecture expands the addressing capabilities and instruction set of its predecessor while ensuring flexibility for the future, and maintaining basic software compatibility with the past.

Designers already familiar with the 8048 or 8049 will be able to take with them the education and experience gained from past designs as ever-increasing system performance demands force them to move on to state-of-the-art products. Newcomers will find the power and regularity of the 8051 instruction set an advantage in streamlining both the learning and design processes.

Microcomputer system designers will appreciate the 8051 as basically a single-chip solution to many problems which previously required board-level computers. Designers of real-time control systems will find the high execution speed, on-chip peripherals, and interrupt capabilities vital in meeting the timing constraints of products previously requiring discrete logic designs. And designers of industrial controllers will be able to convert ladder diagrams directly from tested-and-true TTL or relay-logic designs to microcomputer software, thanks to the unique Boolean processing capabilities.

It has not been the intent of this note to gloss over the difficulty of designing microcomputer-based systems. To be sure, the hardware and software design aspects of any new computer system are nontrivial tasks. However, the system speed, and level of integration of the MCS-51™ microcomputers, the power and flexibility of the instruction set, and the sophisticated assembler and other support products combine to give both the hardware and software designer as much of a head start on the problem as possible.
Using the Intel MCS®-51 Boolean Processing Capabilities

Contents

1. INTRODUCTION ................................................. 10-32

2. BOOLEAN PROCESSOR OPERATION ...................... 10-32
   Processing Elements ............................................. 10-33
   Direct Bit Addressing ......................................... 10-34
   Instruction Set ................................................. 10-39
   Simple Instruction Combinations ............................. 10-40

3. BOOLEAN PROCESSOR APPLICATIONS .................... 10-41
   Design Example #1 — Bit Permutation ...................... 10-42
   Design Example #2 — Software Serial I/O .................. 10-45
   Design Example #3 — Combinatorial Logic Equations ..... 10-46
   Design Example #4 — Automotive Dashboard Functions .... 10-49
   Design Example #5 — Complex Control Functions .......... 10-54
   Additional Functions and Uses ............................... 10-59

4. SUMMARY ........................................................ 10-60

APPENDIX A ...................................................... 10-61
1. INTRODUCTION

The Intel microcontroller family now has three new members: the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS® technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

![Figure 1. 8051 Family Pinout Diagram.](image)

Table 1 summarizes the quantitative differences between the members of the MCS-48™ and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of low-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term “8051” will represent all members of the 8051 Family, unless specifically stated otherwise.

<table>
<thead>
<tr>
<th>EPROM Program Memory</th>
<th>ROM Program Memory</th>
<th>External Program Memory</th>
<th>Program Memory (Int/Max)</th>
<th>Data Memory (Bytes)</th>
<th>Instr. Cycle Time</th>
<th>Input/Output Pins</th>
<th>Interrupt Sources</th>
<th>Reg. Banks</th>
</tr>
</thead>
<tbody>
<tr>
<td>8021</td>
<td></td>
<td></td>
<td>1K 1K</td>
<td>64</td>
<td>10 μSec</td>
<td>21</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8022</td>
<td></td>
<td></td>
<td>2K 2K</td>
<td>64</td>
<td>10 μSec</td>
<td>28</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8048</td>
<td>8048</td>
<td>8035</td>
<td>1K 4K</td>
<td>64</td>
<td>2.5 μSec</td>
<td>27</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8049</td>
<td>8049</td>
<td>8039</td>
<td>2K 4K</td>
<td>128</td>
<td>1.36μSec</td>
<td>27</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>8051</td>
<td>8051</td>
<td>8031</td>
<td>4K 64K</td>
<td>128</td>
<td>1.0 μSec</td>
<td>32</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS-51™ architecture is its complete hardware support for one-bit, or Boolean variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51™ Boolean Processor. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used. It is assumed the reader has read Application Note AP-69, An Introduction to the Intel® MCS-51™ Single-Chip Microcomputer Family, publication number 121518, or has been exposed to Intel's single-chip microcomputer product lines.

For detailed information on these parts refer to the Intel MCS-51™ Family User's Manual, publication number 121517. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the MCS-51™ Macro Assembler User's Guide, publication number 9800937.

2. BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):
a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions;

• a memory to store the sequence of instructions making up a program or algorithm;

• data memory to store variables used by the program; and

• some means of communicating with the outside world.

Figure 2. Block Diagram for Abstract Digital Computer.

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer’s flexibility comes in part from combining simple fast operations to produce more complex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex “virtual” machine.

In fact, any digital computer with the above four functional elements can (given time) complete any algorithm (though the proverbial room full of chimpanzees at word processors might first re-create Shakespeare’s classics and this Application Note!) This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm must proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well- or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the “primitive” operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let’s see how the four basic elements of a digital computer - a CPU with associated registers, program memory, addressable data RAM, and I/O capability - relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instructions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3.a). The more general instructions add a “direct address” byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3.b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.
Table 2. MCS-51™ Boolean Processing Instruction Subset.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETB C</td>
<td>Set Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SETB bit</td>
<td>Set direct Bit</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>CLR C</td>
<td>Clear Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR bit</td>
<td>Clear direct bit</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>CPI C</td>
<td>Complement Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL bit</td>
<td>Complement direct bit</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV C.bit</td>
<td>Move direct bit to Carry flag</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV bit.C</td>
<td>Move Carry flag to direct bit</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL C.bit</td>
<td>AND direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL bit</td>
<td>AND complement of direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL C.bit</td>
<td>OR direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL bit</td>
<td>OR complement of direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JC rel</td>
<td>Jump if Carry flag is set</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC rel</td>
<td>Jump if No Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JB bit.rel</td>
<td>Jump if direct Bit set</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>JNB bit.rel</td>
<td>Jump if direct Bit Not set</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>JBC bit.rel</td>
<td>Jump if direct Bit is set &amp; Clear bit</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Address mode abbreviations:

C — Carry flag.

bit — 128 software flags, any I/O pin, control or status bit

rel — All conditional jumps include an 8-bit offset byte. Range is +127 -128 bytes relative to first byte of the following instruction.

All mnemonics copyrighted© Intel Corporation 1980

Data Memory: The instructions in Figure 3.b can operate directly upon 144 general purpose bits forming the Boolean processor “RAM.” These bits can be used as software flags or to store program variables. Two operand instructions use the CPU’s carry flag (“C”) as a special one-bit register; in a sense, the carry is a “Boolean accumulator” for logical operations and data transfers.

Input/Output: All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer counters, serial port modes, interrupt logic, and so forth).

Figure 3: Bit Addressing Instruction Formats.

Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 4.a). They are numbered consecutively from the lowest-order byte’s lowest-order bit through the highest-order byte’s highest-order bit.

Bit addresses between 128 and 255(80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register’s own address, while the three low-order bits identify the bit position within that register (Figure 4.b).
Notice the column labeled “Symbol” in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C. P0, P1, P2, and P3 are the 8051’s four I/O ports; secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses, the five bits not implemented in IE and IP should not be accessed; they can not be used as software flags.
### (MSB) Symbol Position | Name and significance
---|---
CY | PSW.7 Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC | PSW.6 Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0 | PSW.5 Flag 0. Set/cleared/tested by software as a user-defined status flag.
RS1 | PSW.4 Register bank Select control bits
RS0 | PSW.3 1 & 0. Set/cleared by software to determine working register bank (see Note)

### (LSB) Symbol Position | Name and significance
---|---
OV | PSW.2 Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
— | PSW.1 (reserved)
P | PSW.0 Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of “one” bits in the accumulator, i.e., even parity.

**Note** - the contents of (RS1, RS0) enable the working register banks as follows:

- (0,0) - Bank 0 (00H-07H)
- (0,1) - Bank 1 (08H-0FH)
- (1,0) - Bank 2 (10H-17H)
- (1,1) - Bank 3 (18H-1FH)

---

**Figure 5. PSW - Program Status Word organization.**

### (MSB) Symbol Position | Name and significance
---|---
RD | P3.7 Read data control output. Active low pulse generated by hardware when external data memory is read.
WR | P3.6 Write data control output. Active low pulse generated by hardware when external data memory is written.
T1 | P3.5 Timer/counter 1 external input or test pin.
T0 | P3.4 Timer/counter 0 external input or test pin.

### (LSB) Symbol Position | Name and significance
---|---
INT1 | P3.3 Interrupt 1 input pin. Low-level or falling-edge triggered.
INT0 | P3.2 Interrupt 0 input pin. Low-level or falling-edge triggered.
TXD | P3.1 Transmit Data pin for serial port in UART mode. Clock output in shift register mode.
RXD | P3.0 Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

**Figure 6. P3 - Alternate I/O Functions of Port 3.**
Symbol Position | Name and significance
---|---
TF1 | TCON.7 Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR1 | TCON.6 Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
TF0 | TCON.5 Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR0 | TCON.4 Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.

a.) TCON - Timer/Counter Control/status register.

Symbol Position | Name and significance
---|---
SM0 | SCON.7 Serial port Mode control bit 0. Set/cleared by software (see note).
SM1 | SCON.6 Serial port Mode control bit 1. Set/cleared by software (see note).
SM2 | SCON.5 Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.
REN | SCON.4 Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.
TB8 | SCON.3 Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.

b.) SCON - Serial Port Control/status register.

IE1 | TCON.3 Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1 | TCON.2 Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
IE0 | TCON.1 Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0 | TCON.0 Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

Figure 7. Peripheral Configuration Registers.
(MSB) | (LSB)
---|---
EA | — | — | ES | ET1 | EX1 | ET0 | EXO | EX0

Symbol Position | Name and significance
---|---
EA | IE 7 | Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4 - IE.0.
— | IE.6 | (reserved)
— | IE.5 | —
ES | IE 4 | Enable Serial port control bit Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1 | IE 3 | Enable Timer 1 control bit Set/cleared by software to enable/disable interrupts from timer/counter 1

Ex1 | IE.2 | Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0 | IE.1 | Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0.
EX0 | IE 0 | Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

c.) IE - Interrupt Enable Register.

(MSB) | (LSB)
---|---
— | — | — | PS | PT1 | PX1 | PT0 | PX0

Symbol Position | Name and significance
---|---
— | IP.7 | (reserved)
— | IP.6 | (reserved)
— | IP.5 | (reserved)
PS | IP.4 | Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1 | IP.3 | Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.

PX1 | IP.2 | External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0 | IP 1 | Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0 | IP 0 | External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

d.) IP - Interrupt Priority Control Register.

Addressable Register Set. There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51™ architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all non-bit addressable registers and RAM.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.
Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: I or 0. Single byte versions perform the same three operations on the carry.

The MCS-80 assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0-255);
- by the name or address of the register containing the bit, the dot operator symbol (a period: "."), and the bit's position in the register (7.0);
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5-7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

<table>
<thead>
<tr>
<th>USR.Flags BIT</th>
<th>PSW.5</th>
<th>Absolute Addressing</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR 0D5H</td>
<td></td>
<td>Use of Dot Operator</td>
</tr>
<tr>
<td>CLR PSW.5</td>
<td></td>
<td>Pre-Defined Assembler</td>
</tr>
<tr>
<td>CLR USR.Flags</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Transfers. The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

Logical Operations. Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a slash mark ("/") before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit, rel" and "JNB bit,rel" (Jump on Bit and Jump on Not Bit) test the state of any addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit,rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48 "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a pro-
grammer would have quite a chore trying to compute relative offset values from one instruction to another. ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

(The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C.bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does not constitute abstract bit-addressing in the MCS-51™ sense. A flag exists only as a field within a register; to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.)

*Interaction with Other Instructions.* The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

**Simple Instruction Combinations**

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer/counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register. Figure 7.a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48® family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear/complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB,}

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A, Rn</td>
<td>Add register to Accumulator</td>
<td>1 1</td>
</tr>
<tr>
<td>ADD A, direct</td>
<td>Add direct byte to Accumulator</td>
<td>1 1</td>
</tr>
<tr>
<td>ADD A, @Ri</td>
<td>Add indirect RAM to Accumulator</td>
<td>1 1</td>
</tr>
<tr>
<td>ADD A, #data</td>
<td>Add immediate data to Accumulator</td>
<td>1 1</td>
</tr>
<tr>
<td>ADDC A, Rn</td>
<td>Add register to Accumulator with Carry flag</td>
<td>1 1</td>
</tr>
<tr>
<td>ADDC A, direct</td>
<td>Add direct byte to Accumulator with Carry flag</td>
<td>1 1</td>
</tr>
<tr>
<td>ADDC A, @Ri</td>
<td>Add indirect RAM to Accumulator with Carry flag</td>
<td>2 1</td>
</tr>
<tr>
<td>ADDC A, #data</td>
<td>Add immediate data to Accumulator with borrow</td>
<td>2 1</td>
</tr>
<tr>
<td>SUBB A, Rn</td>
<td>Subtract register from Accumulator with borrow</td>
<td>2 1</td>
</tr>
<tr>
<td>SUBB A, direct</td>
<td>Subtract direct byte from Acc with borrow</td>
<td>2 1</td>
</tr>
<tr>
<td>SUBB A, @Ri</td>
<td>Subtract indirect RAM from Acc with borrow</td>
<td>2 1</td>
</tr>
<tr>
<td>SUBB A, #data</td>
<td>Subtract immediate data from Acc with borrow</td>
<td>2 1</td>
</tr>
<tr>
<td>MUL AB</td>
<td>Multiply A &amp; B</td>
<td>1 4</td>
</tr>
<tr>
<td>DIV AB</td>
<td>Divide A by B</td>
<td>1 4</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal Adjust Accumulator</td>
<td>1 1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate Accumulator Left through the Carry flag</td>
<td>1 1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate Accumulator Right through Carry flag</td>
<td>1 1</td>
</tr>
<tr>
<td>CJNE A, direct, rel</td>
<td>Compare direct byte to Acc &amp; Jump if Not Equal</td>
<td>3 2</td>
</tr>
<tr>
<td>CJNE A, #data, rel</td>
<td>Compare immediate to Acc &amp; Jump if Not Equal</td>
<td>3 2</td>
</tr>
<tr>
<td>CJNE Rn, #data, rel</td>
<td>Compare immem to register &amp; Jump if Not Equal</td>
<td>3 2</td>
</tr>
<tr>
<td>CJNE @Ri, #data, rel</td>
<td>Compare immem to indirect &amp; Jump if Not Equal</td>
<td>3 2</td>
</tr>
</tbody>
</table>

All mnemonics copyrighted © Intel Corporation 1980

10-40
CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 4.a shows how 8051 programs implement software flag and machine control functions associated with special using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time. Quantitatively, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to microcontroller applications.

The opcode space freed by condensing many specific 8048

<table>
<thead>
<tr>
<th>8048 Instruction</th>
<th>Bytes</th>
<th>Cycles</th>
<th>uSec</th>
<th>8x51 Instruction</th>
<th>Bytes</th>
<th>Cycles &amp; uSec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flag Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR C</td>
<td>1</td>
<td>1</td>
<td>2.5</td>
<td>CLR C</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td>1</td>
<td>1</td>
<td>2.5</td>
<td>CPL F0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Flag Testing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JNC offset</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td>JNC rel</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF0 offset</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td>JB F0,rel</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>JB7 offset</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td>JB ACC.7,rel</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td><strong>Peripheral Polling</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JTO offset</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td>JB T0,rel</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>JNI offset</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td>JNB INTO,rel</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>JTF offset</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td>JBC TF0,rel</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td><strong>Machine and Peripheral Control</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRT T</td>
<td>1</td>
<td>1</td>
<td>2.5</td>
<td>SETB TR0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>EN I</td>
<td>1</td>
<td>1</td>
<td>2.5</td>
<td>SETB EX0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>DIS TCNTI</td>
<td>1</td>
<td>1</td>
<td>2.5</td>
<td>CLR ET0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.b. Replacing 8048 instruction sequences with single 8x51 instructions.

<table>
<thead>
<tr>
<th>8048 Instructions</th>
<th>Bytes</th>
<th>Cycles</th>
<th>uSec</th>
<th>8051 Instructions</th>
<th>Bytes</th>
<th>Cycles &amp; uSec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flag Control</strong></td>
<td></td>
<td></td>
<td></td>
<td>SETB C</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Set carry:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR C</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPL C</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Software Flag:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR F0</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPL F0</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

opcodes in the 8048. In every case the MCS-51™ solution requires the same number of machine cycles, and executes 2.5 times faster.

3. BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

Qualitatively, nothing. All the same capabilities could be (and often have been) implemented on other machines instructions into a few general operations has been used to add new functionality to the MCS-51™ architecture - both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4.b. Here the 8051 speed advantage ranges from 5x to 15x!
### Table 4b (Continued)

<table>
<thead>
<tr>
<th>8048 Instructions</th>
<th>Bytes</th>
<th>Cycles</th>
<th>uSec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn Off Output Pin: ANL P1.0, #0FBH</td>
<td>2</td>
<td>2</td>
<td>5.0</td>
</tr>
<tr>
<td>Complement Output Pin: IN A, P1 XRL A, #04H OUTL P1.A</td>
<td>4</td>
<td>6</td>
<td>15.0</td>
</tr>
<tr>
<td>Clear Flag in RAM: MOV R0, #FLGADR MOV A, @R0 ANL A, #FLGMASK MOV @R0.A</td>
<td>6</td>
<td>6</td>
<td>15.0</td>
</tr>
<tr>
<td>Flag Testing: Jump if Software Flag is 0: JF0 $+4 JMP offset</td>
<td>4</td>
<td>4</td>
<td>10.0</td>
</tr>
<tr>
<td>Jump if Accumulator bit is 0: CPL A JB7 offset CPL A</td>
<td>4</td>
<td>4</td>
<td>10.0</td>
</tr>
<tr>
<td>Peripheral Polling Test if Input Pin is Grounded: IN A, P1 CPL A JB3 offset</td>
<td>4</td>
<td>5</td>
<td>12.5</td>
</tr>
<tr>
<td>Test if Interrupt Pin is High: JNI $+4 JMP offset</td>
<td>4</td>
<td>4</td>
<td>10.0</td>
</tr>
<tr>
<td>8x51 Instructions</td>
<td>Bytes</td>
<td>Cycles &amp; uSec</td>
<td></td>
</tr>
<tr>
<td>CLR P1.2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CPL P1.2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CLR USER_FLG</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>JNB F0, rel</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JNB ACC.7, rel</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JNB P1.3, rel</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>JB INTO, rel</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51™ based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved;
- easier to debug because more individual instructions have no unexpected or undesirable side-effects;
- more byte efficient due to direct bit addressing and program counter relative branching;
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed;
- lower cost because of the high level of system-interrogation within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first three compare the 8051 with other microprocessors; the last two go into 8051-based system designs in much greater depth.

**Design Example #1 - Bit Permutation**

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing Data Encryption Standard (DES).

Basically, the DES combines eight bytes of “plaintext” data (in binary, ASCII, or any other format) with a 56-bit “key”, producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.
It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive-OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of Electronics magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

![Figure 9. DES Key Schedule Transformation.](image)

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10.a:

- Initialize the Permutation Buffer to default state (ones or zeroes);
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte;
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct;
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving
them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB_1, SKB_2, ..., SKB_56, and that the bytes of the latter are named PB_1, ..., PB_8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1.a. The total routine length would be 192 bytes, requiring 144 microseconds.

The algorithm of Figure 10.b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 usec.) and shifting it into the accumulator (1 usec.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1.b.

**Example 1. DES Key Permutation Software.**

a.) “Brute Force” technique.

```
MOV C,SKB_1
MOV PB_1.1,C
MOV C,SKB_2
MOV PB_4.0,C
MOV C,SKB_3
MOV PB_2.5,C
MOV C,SKB_4
MOV PB_1.0,C
...
...
MOV C,SKB_55
MOV PB_5.0,C
MOV C,SKB_56
MOV PB_7.2,C
```

b.) Using Accumulator to Collect Bits.

```
CLR A
MOV C,SKB_14
RLC A
MOV C,SKB_17
RLC A
MOV C,SKB_11
RLC A
MOV C,SKB_24
RLC A
MOV C,SKB_1
RLC A
MOV C,SKB_5
RLC A
MOV PB_1,A
...
...
MOV C,SKB_29
RLC A
MOV C,SKB_32
RLC A
MOV PB_8,A
```

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 usec.)

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11.a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11.b). The whole DES algorithm would require less than one-
fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

---

**Figure 11. Secure Banking Terminal Block Diagram.**

**Design Example #2 - Software Serial I/O**

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. (See, for example, Application Notes AP24, AP29, and AP49.) Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figures 12.a and 12.b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

---

**Figure 12. Serial I/O Algorithms.**

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 5.a and 5.b. The 8051 solution is more efficient than the others on every count!
### Table 5. Serial I/O Programs for Various Microprocessors.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Code</th>
<th>Code</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>8085</td>
<td>8048</td>
<td>8051</td>
<td></td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td><strong>ANI</strong></td>
<td><strong>CLR C</strong></td>
<td><strong>MOV C, SERPIN</strong></td>
</tr>
<tr>
<td><strong>SERPORT</strong></td>
<td><strong>MASK</strong></td>
<td><strong>C</strong></td>
<td><strong>C</strong></td>
</tr>
<tr>
<td><strong>JZ</strong></td>
<td><strong>JNTO</strong></td>
<td><strong>LO</strong></td>
<td><strong>LO</strong></td>
</tr>
<tr>
<td><strong>CMC</strong></td>
<td><strong>CPL C</strong></td>
<td><strong>CMC</strong></td>
<td><strong>CMC</strong></td>
</tr>
<tr>
<td><strong>LXH, SERBUF</strong></td>
<td><strong>MOV R0, #SERBUF</strong></td>
<td><strong>MOV A, @R0</strong></td>
<td><strong>MOV A, SERBUF</strong></td>
</tr>
<tr>
<td><strong>MOV A, M</strong></td>
<td><strong>MOV A, @R0</strong></td>
<td><strong>RRC A</strong></td>
<td><strong>RRC A</strong></td>
</tr>
<tr>
<td><strong>RR</strong></td>
<td><strong>RRC A</strong></td>
<td><strong>RRC A</strong></td>
<td><strong>RRC A</strong></td>
</tr>
<tr>
<td><strong>MOV M, A</strong></td>
<td><strong>MOV @R0, A</strong></td>
<td><strong>MOV SERBUF, A</strong></td>
<td><strong>MOV SERBUF, A</strong></td>
</tr>
</tbody>
</table>

RESULTS:

- **8085**: 8 INSTRUCTIONS, 14 BYTES, 56 STATES, 19 uSEC.
- **8048**: 8 INSTRUCTIONS, 13 BYTES, 72 STATES, 27.5 uSEC.
- **8051**: 4 INSTRUCTIONS, 7 BYTES, 4 CYCLES, 4 uSEC.

### b. Output Routine.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Code</th>
<th>Code</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>8085</td>
<td>8048</td>
<td>8051</td>
<td></td>
</tr>
<tr>
<td><strong>LXI HL, SERBUF</strong></td>
<td><strong>MOV R0, #SERBUF</strong></td>
<td><strong>MOV A, SERBUF</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MOV A, M</strong></td>
<td><strong>MOV A, @R0</strong></td>
<td><strong>RRC A</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RR</strong></td>
<td><strong>RRC A</strong></td>
<td><strong>RRC A</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MOV M, A</strong></td>
<td><strong>MOV @R0, A</strong></td>
<td><strong>MOV SERBUF, A</strong></td>
<td></td>
</tr>
<tr>
<td><strong>IN SERPORT</strong></td>
<td><strong>JC HI</strong></td>
<td><strong>JC HI</strong></td>
<td></td>
</tr>
<tr>
<td><strong>JMP CNT</strong></td>
<td><strong>JMP CNT</strong></td>
<td><strong>JMP CNT</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ORI MASK</strong></td>
<td><strong>ORL SERPRT, #MASK</strong></td>
<td><strong>MOV SERPIN, C</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CNT OUT SERPORT</strong></td>
<td><strong>CNT</strong></td>
<td><strong>CNT</strong></td>
<td></td>
</tr>
</tbody>
</table>

RESULTS:

- **8085**: 10 INSTRUCTIONS, 20 BYTES, 72 STATES, 24 uSEC.
- **8048**: 8 INSTRUCTIONS, 13 BYTES, 11 CYCLES, 27.5 uSEC.
- **8051**: 4 INSTRUCTIONS, 7 BYTES, 5 CYCLES, 5 uSEC.

### Design Example #3 - Combinatorial Logic Equations

Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

\[
Q = (U \cdot (V + W)) \cdot (X \cdot \overline{Y}) + \overline{Z}
\]

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.
a.) Using TTL:

For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51™ instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the “True” or “Not True” exit ASAP — as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won’t use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51™ mnemonics are used in Example 2.a; other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move/mask/conditional jump sequence in Example 2.a, but the algorithm would be equally convoluted (see Example 2.B). To lessen the confusion “a bit” each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2.c) strings together the Boolean ANL and ORL functions to generate the output function with straight-line code.
When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N+1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles," perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions.

;BFUNC1 SOLVE RANDOM LOGIC FUNCTION OF 6 VARIABLES BY DIRECTLY LOADING AND MASKING THE APPROPRIATE BITS IN THE ACCUMULATOR, THEN EXECUTING CONDITIONAL JUMPS BASED ON ZERO CONDITION. (APPROACH USED BY BYTE-ORTIENED ARCHITECTURES.) BYTE AND MASK VALUES CORRESPOND TO RESPECTIVE BYTE ADDRESS AND BIT POSITIONS.

OUTBUF DATA 22H ;OUTPUT PIN STATE MAP

TESTV: MOV A,P2
ANL A,#00000100B
JNZ TESTU
MOV A,TCON
ANL A,#00100000B
JZ TESTX

TESTU: MOV A,P1
ANL A,#00000010B
JNZ SETQ

TESTX: MOV A,TCON
ANL A,#00001000B
JZ TESTZ
MOV A,20H
ANL A,#00000001B
JZ SETQ

TESTZ: MOV A,21H
ANL A,#00000010B
JZ SETQ

CLRQ: MOV A,OUTBUF
ANL A,#11110111B
JMP OUTQ

SETQ: MOV A,OUTBUF
ORL A,#00001000B

OUTQ: MOV OUTBUF,A
MOV P3,A

b.) Using only bit-test instructions.

;BFUNC2 SOLVE A RANDOM LOGIC FUNCTION OF 6 VARIABLES BY DIRECTLY POLLING EACH BIT. (APPROACH USING MCS-51 UNIQUE BIT-TEST INSTRUCTION CAPABILITY.) SYMBOLS USED IN LOGIC DIAGRAM ASSIGNED TO CORRESPONDING 8x51 BIT ADDRESSES.

U BIT P1.1
V BIT P2.2
W BIT TF0
X BIT IE1
Y BIT 20H.0
Z BIT 21H.1
Q BIT P3.3

TEST_V: JB V.TEST_U
JNB W.TEST_X
TEST_U: JB U.SET_Q
TEST_X: JNB X.TEST_Z
JNB Y.SET_Q
TEST_Z: JNB Z.SET_Q
CLR_Q: CLR Q
JMP NXTTST
SET_Q: SETB Q

NXTTST: ; (CONTINUATION OF PROGRAM)

C.) Using logical operations on Boolean variables.

;FUNC3 SOLVE A RANDOM LOGIC FUNCTION OF 6 VARIABLES USING STRAIGHT-LINE LOGICAL INSTRUCTIONS ON MCS-51 BOOLEAN VARIABLES.

MOV C.V
ORI C.W ;OUTPUT OF OR GATE
ANL C.U ;OUTPUT OF TOP AND GATE
MOV F0,C ;SAVE INTERMEDIATE STATE
MOV C.X
ANL C.Y ;OUTPUT OF BOTTOM AND GATE
ORI C.F0 ;INCLUDE VALUE SAVED ABOVE
ORI C.Z ;INCLUDE LAST INPUT VARIABLE
MOV Q.C ;OUTPUT COMPUTED RESULT
An upper-limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The actual total should be somewhat shorter, since calculations can be "chained," as shown above. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

**Design Example #4 - Automotive Dashboard Functions**

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the three position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly . . . unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

![Figure 15. TTL logic implementation of automotive turn signals.](image-url)

**Table 6. Truth table for turn-signal operation.**

<table>
<thead>
<tr>
<th>BRAKE SWITCH</th>
<th>EMERG. SWITCH</th>
<th>LEFT FRONT TURN SWITCH</th>
<th>RIGHT TURN SWITCH</th>
<th>LEFT FRONT &amp; DASH</th>
<th>RIGHT FRONT &amp; DASH</th>
<th>LEFT REAR</th>
<th>RIGHT REAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>OFF</td>
<td>BLINK</td>
<td>OFF</td>
<td>BLINK</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>BLINK</td>
<td>BLINK</td>
<td>BLINK</td>
<td>BLINK</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>OFF</td>
<td>BLINK</td>
<td>ON</td>
<td>BLINK</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>BLINK</td>
<td>BLINK</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>BLINK</td>
<td>BLINK</td>
<td>ON</td>
<td>BLINK</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>BLINK</td>
<td>BLINK</td>
<td>ON</td>
<td>BLINK</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>BLINK</td>
<td>BLINK</td>
<td>ON</td>
<td>BLINK</td>
</tr>
</tbody>
</table>

0149A-20

10-49
In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

The Single-chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

### INPUT PIN DECLARATIONS:

(All inputs are positive-true logic)

- BRAKE BIT P1.0: BRAKE PEDAL DEPRESSED
- EMERG BIT P1.1: EMERGENCY BLINKER ACTIVATED
- PARK BIT P1.2: PARKING LIGHTS ON
- L_TURN BIT P1.3: TURN LEVER DOWN
- R_TURN BIT P1.4: TURN LEVER UP

### OUTPUT PIN DECLARATIONS:

- L_FRNT BIT P1.5: FRONT LEFT-TURN INDICATOR
- R_FRNT BIT P1.6: FRONT RIGHT-TURN INDICATOR
- L_DASH BIT P1.7: DASHBOARD LEFT-TURN INDICATOR
- R_DASH BIT P2.0: DASHBOARD RIGHT-TURN INDICATOR
- L_REAR BIT P2.1: REAR LEFT-TURN INDICATOR
- R_REAR BIT P2.2: REAR RIGHT-TURN INDICATOR

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It’s quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use “clever” algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevant, and masks and complex program structures are not needed. Only the initial Boolean varia-
Timer 0 (one of the two on-chip timer/counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TL0) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256 µSec. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows. (For details of the numerous timer operating modes see the MCS-51® User’s Manual.)

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very fast to modulate the parking lights; bit 7 will be “tuned” to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4.096 msec. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB_DIV. Loading SUB_DIV. with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

The code to sample inputs, perform calculations, and update outputs—the real “meat” of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm \((\text{PARK} \cdot \text{H_FREQ})\), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named “DIM”. The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since The PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, independent of...
the hardware configuration. The fact that these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```
MOV C.L_TURN: SET CARRY IF TURN
ORI. C.EMERG: OR EMERGENCY SELECTED.
ANI. C.IO_FREQ: GATE IN 1 Hz SIGNAL.
MOV L_DASH.C: AND OUTPUT TO DASHBOARD.
```

To generate the left front turn signal we only need to add the parking light function in F0. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in FO.

```
MOV F0.C: SAVE FUNCTION SO FAR.
ORI. C.DIM: ADD IN PARKING LIGHT FUNCTION
MOV L_FRNT.C: AND OUTPUT TO TURN SIGNAL.
```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```
MOV C.BRAKE: GATE BRAKE PEDAL SWITCH
ANI. C.L_TURN: WITH TURN LEVER.
ORI. C.F0: INCLUDE TEMP. VARIABLE FROM DASH
ORI. C.DIM: AND PARKING LIGHT FUNCTION
MOV L_REAR.C: AND OUTPUT TO TURN SIGNAL.
```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```
MOV C.R_TURN: SET CARRY IF TURN
ORI. C.EMERG: OR EMERGENCY SELECTED.
ANI. C.IO_FREQ: IF SO, GATE IN 1 Hz SIGNAL.
```

```
MOV R_DASH.C: AND OUTPUT TO DASHBOARD.
MOV F0.C: SAVE FUNCTION SO FAR.
ORI. C.DIM: ADD IN PARKING LIGHT FUNCTION
MOV R_FRNT.C: AND OUTPUT TO TURN SIGNAL.
MOV C.BRAKE: GATE BRAKE PEDAL SWITCH
ANI. C.R_TURN: WITH TURN LEVER.
ORI. C.F0: INCLUDE TEMP. VARIABLE FROM DASH
ORI. C.DIM: AND PARKING LIGHT FUNCTION
MOV R_REAR.C: AND OUTPUT TO TURN SIGNAL.
```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

```
POP B: RESTORE CPU REGISTERS.
POP ACC
POP PSW
RETI
```

Program Refinements. The luminescence of an incandescent light bulb filament is generally non-linear: the 50% duty cycle of HL_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB_DIV. For example, 30 Hz signals of seven different duty cycles could be produced by considering bits 2—0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM:

```
MOV C.SUB_DIV.1: START WITH 50 PERCENT
ANI. C.SUB_DIV.0: MASK DOWN TO 25 PERCENT
ORI. C.SUB_DIV.2: AND BUILD BACK TO 62 PERCENT
MOV DIM.C: DUTY CYCLE FOR PARKING LIGHTS.
```
Table 7. Non-trivial Duty Cycles.

<table>
<thead>
<tr>
<th>SUB_DIV BITS</th>
<th>DUTY CYCLES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12.5%</td>
</tr>
<tr>
<td>X X X X X X 0 0 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 0 0 1</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 0 0 0 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 0 1 0 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 0 1 0 1 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 0 1 0 1 1 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 1 0 0 0 0 0 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 1 0 0 0 1 0 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 1 0 0 1 0 0 0 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 0 1 0 0 1 0 0 1 0</td>
<td>OFF</td>
</tr>
<tr>
<td>X X X X X X 1 0 0 0 0 0 0 0 0</td>
<td>ON</td>
</tr>
<tr>
<td>X X X X X X 1 0 0 0 1 0 0 0</td>
<td>ON</td>
</tr>
</tbody>
</table>

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the “RXD” pin while a burst of eight clock pulses is generated on the “TXD” pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 usec. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn’t notice. Some shift registers provide an “enable” bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

L_FRNT BIT B.0 : FRONT LEFT-TURN INDICATOR
R_FRNT BIT B.1 : FRONT RIGHT-TURN INDICATOR
L_DASH BIT B.2 : DASHBOARD LEFT-TURN INDICATOR
R_DASH BIT B.3 : DASHBOARD RIGHT-TURN INDICATOR

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port:

MOV SBUF.B : LOAD BUFFER AND TRANSMIT

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and or fault detection on the four main turn indicators. Each could still be a standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing
circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

Assume all of the lights are turned on except one; i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12 V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12 V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

Now turn all the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12 V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB_DIV is reloaded by the interrupt routine.

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidently) 67 usec., plus 32 usec once per second for the electrical test. If executed every 4 msec as suggested this software would typically reduce the throughput of the background program by less than 2%. Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation — true flash factor, so to speak.

**Design Example #5 - Complex Control Functions**

Finally, we’ll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.
Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors;
- 12 output signals;
- Combinational and sequential logic computations;
- Remote operation with communications to a host processor via a high-speed full-duplex serial line;
- Two prioritized external interrupts;
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs no other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix; as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

The computer’s serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20.a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kohm resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0 V logic threshold, even in the worst-case, where all contacts in an enabled column are closed.) Since PO is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

The circuits in Figures 20.b—20.d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs and provide optical isolation as in Figure 20.b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20.c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20.d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and/or debounce contact closures by comparing each bit with its earlier value.

Figure 19. Block diagram of 64-input machine controller.
The code in Example 3 implements the scanning algorithm for the circuits in Figure 20.a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic: ones represent contacts that are closed or isolators turned on.
MOV A,#80H: SET FIRST BIT IN ACC.
SCAN: MOV P1.A: OUTPUT TO SCAN LINES.
RR A: SHIFT TO ENABLE NEXT COLUMN NEXT.
MOV R2,A: REMEMBER CURRENT SCAN POSITION.
MOV A,P0: READ RETURN LINES.
XCH A,@R0: SWITCH WITH PREVIOUS MAP BITS.
MOV @R1.A: SAVE PREVIOUS STATE AS WELL.
INC R0: BUMP POINTERS.
INC R1: 
MOV A,R2: RELOAD SCAN LINE MASK
JNB ACC.7,SCAN: LOOP UNTIL ALL EIGHT COLUMNS READ.
RET

Figure 21. Flowchart for reading in sensor matrix.

What happens after the sensors have been scanned depends on the individual application. Rather than inventing some artificial design problem, software corresponding to commonplace logic elements will be discussed.

**Combinatorial Output Variables.** An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps; in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.

Simple Combinatorial Output Variables.

: SET P2.2 = (12) (23) (34) (45) (56)
  MOV C.12
  ANI. C.23
  ANI. C.34
  ANI. C.45
  ANI. C.56
  MOV P2.2,C

**Intermediate Variables.** The examination of a typical relay-logic ladder diagram will show that many of the rungs control not outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51™ solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and saved (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

CALL. INPUT_SCAN
  MOV C.0
  ORI. C.1
  ORI. C.2
  ORI. C.3
  MOV F0.C
COMPUTE FUNCTION 0
ANI. C. F0
MOV P1.0.C
...

COMPUTE FUNCTION 1
ANI. C. F0
MOV P1.1.C
...

COMPUTE FUNCTION 2
ANI. C. F0
MOV P1.2.C
...

Figure 22. Ladder diagram for output override circuitry.

Latching Relays. A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state—analogous to a TTL Set/Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an "emergency cleared" button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

Example 6: Simulating a latching relay.

:L_SET   SET FLAG.0 IF C=1
L_SET:   ORI. C.F0
          MOV F0.C

:L_RESET RESET FLAG 0 IF C=1
L_RESET: CPS C,
          ANI. C.F0
          MOV F0.C

Time Delay Relays. A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer/counters. The procedure followed by the routine depends heavily on the details of the exact function needed: timeouts or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

Example 7. Code to clear USRFLG after a fixed time delay.

JNB  USR_FLG,NXTTST
DJNZ DLAY_COUNT,NXTTST
CLR USR_FLG
MOV DLAY_COUNT,#200
NXTTST: ...
...

Serial Interface to Remote Processor. When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shutdown sequence each should initiate. For more information on using the serial port, consult the MCS-51™ User's Manual.

Response Timing.
One difference between relay and programmed industrial controllers (when each is considered as a "black box") is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a
A large number of “rungs” operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. Every instruction mentioned in this Note completes in one or two microseconds—the minimum instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond; less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

Additional functions and uses.

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

```
: EXCLUSIVE-OR FUNCTION IMPOSED ON CARRY
: USING F0 IS INPUT VARIABLE.
XOR_F0: JNB F0,XORCNT ; ( "JB" FOR X-NOR)
  CPL  C
XORCNT: ....
```

`XCH. The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-through-carry instructions, though this would alter the accumulator data.```

```
: EXCHANGE CARRY WITH USRFLG
XCHBIT: RLC  A
  MOV  C,USR_FLG
  RRC  A
  MOV  USR_FLG,C
  RLC  A
```

Extended Bit Addressing. The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3.b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

Parity of bytes or bits. The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes.

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much
longer bit streams. The algorithms presented in Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers. Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

4. SUMMARY
A truly unique facet of the Intel MCS-51™ microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I/O options. These are the world’s first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some ways from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I/O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.
THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET
OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER
OF AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO
TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT
CONTROL, AND PARKING LIGHT OPERATION.
THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN
EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70,
"USING THE INTEL MCS-51(TM)
BOOLEAN PROCESSING CAPABILITIES"

INPUT PIN DECLARATIONS:
(ALL INPUTS ARE POSITIVE-TRUE LOGIC.
INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED )
BIT PI. 0  BRAKE PEDAL DEPRESSED
BIT PI. 1  EMERGENCY BLINKER ACTIVATED
BIT PI. 2  PARKING LIGHTS ON
BIT PI. 3  TURN LEVER DOWN
BIT PI. 4  TURN LEVER UP

OUTPUT PIN DECLARATIONS:
(ALL OUTPUTS ARE POSITIVE TRUE LOGIC
BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)
BIT P1 5  FRONT LEFT-TURN INDICATOR
BIT P1 6  FRONT RIGHT-TURN INDICATOR
BIT P2 7  DASHBOARD LEFT-TURN INDICATOR
BIT P2 8  DASHBOARD RIGHT-TURN INDICATOR
BIT P2 9  REAR LEFT-TURN INDICATOR
BIT P2 0  REAR RIGHT-TURN INDICATOR

INTERNAL VARIABLE DEFINITIONS:
BIT SUB_DIV 20H  INTERRUPT RATE SUBDIVIDER
BIT SUB_DIV 0  HIGH-FREQUENCY OSCILLATOR BIT
BIT SUB_DIV 7  LOW-FREQUENCY OSCILLATOR BIT
BIT PSW 1  PARKING LIGHTS ON FLAG

*****************************************************************************
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>020040</td>
<td>49</td>
<td>ORG 0000H, RESET VECTOR</td>
</tr>
<tr>
<td>0000</td>
<td>020040</td>
<td>50</td>
<td>LJMP INIT</td>
</tr>
<tr>
<td>000B</td>
<td>75BCF0</td>
<td>51</td>
<td>ORG 000BH, TIMER 0 SERVICE VECTOR</td>
</tr>
<tr>
<td>000B</td>
<td>75BCF0</td>
<td>52</td>
<td>MOV TMO, #16, HIGH TIMER BYTE ADJUSTED TO CONTROL INT RATE</td>
</tr>
<tr>
<td>000E</td>
<td>CDDO</td>
<td>53</td>
<td>PUSH PSW, EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW</td>
</tr>
<tr>
<td>0010</td>
<td>0154</td>
<td>54</td>
<td>LJMP UPDATE, (CONTINUE WITH REST OF ROUTINE)</td>
</tr>
<tr>
<td>0040</td>
<td>75BAC0</td>
<td>55</td>
<td>MOV TLO, #16, ZERO LOADED INTO LOW-ORDER BYTE AND</td>
</tr>
<tr>
<td>0043</td>
<td>75BCF0</td>
<td>56</td>
<td>MOV TMOD, #01H, 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,</td>
</tr>
<tr>
<td>0046</td>
<td>75B961</td>
<td>57</td>
<td>MOV TMOD, #0110001B, 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,</td>
</tr>
<tr>
<td>0049</td>
<td>7520F4</td>
<td>58</td>
<td>MOV SUB_DIV, #244, SUBDIVIDE INTERRUPT RATE BY 244 FOR 1HZ</td>
</tr>
<tr>
<td>004C</td>
<td>D2A9</td>
<td>59</td>
<td>SETB EA, CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS</td>
</tr>
<tr>
<td>004E</td>
<td>D2AF</td>
<td>60</td>
<td>SETB TRO, KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW</td>
</tr>
<tr>
<td>0050</td>
<td>D2BC</td>
<td>61</td>
<td>MOV TMOD, #110016, 16-BIT TIMER MODE FOR TIMER 0 SELECTED</td>
</tr>
<tr>
<td>0052</td>
<td>005E</td>
<td>62</td>
<td>MOV TMOD, #11100001B, 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,</td>
</tr>
<tr>
<td>0054</td>
<td>D52038</td>
<td>63</td>
<td>MOV TMOD, #1000111B, 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,</td>
</tr>
<tr>
<td>0057</td>
<td>7520F4</td>
<td>64</td>
<td>MOV TMOD, #11100001B, 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,</td>
</tr>
<tr>
<td>005A</td>
<td>4390E0</td>
<td>65</td>
<td>ORL P1, #11110000B, SET CONTROL OUTPUTS HIGH</td>
</tr>
<tr>
<td>005D</td>
<td>43A007</td>
<td>66</td>
<td>ORL P2, #00000111B, SET CONTROL OUTPUTS HIGH</td>
</tr>
<tr>
<td>0060</td>
<td>C295</td>
<td>67</td>
<td>CLR L_FRNT, FLOAT DRIVE COLLECTOR</td>
</tr>
<tr>
<td>0062</td>
<td>208428</td>
<td>68</td>
<td>JB TO_FAULT, TO SHOULD BE PULLED LOW</td>
</tr>
<tr>
<td>0065</td>
<td>D295</td>
<td>69</td>
<td>SETB L_FRNT, PULL COLLECTOR BACK DOWN</td>
</tr>
<tr>
<td>0067</td>
<td>C297</td>
<td>70</td>
<td>CLR L_DASH, REPEAT SEQUENCE FOR L_DASH,</td>
</tr>
<tr>
<td>0069</td>
<td>208421</td>
<td>71</td>
<td>JB TO_FAULT,</td>
</tr>
<tr>
<td>006C</td>
<td>D297</td>
<td>72</td>
<td>SETB L_DASH,</td>
</tr>
<tr>
<td>006E</td>
<td>C2A1</td>
<td>73</td>
<td>CLR L_REAR,</td>
</tr>
<tr>
<td>0070</td>
<td>20841A</td>
<td>74</td>
<td>JB TO_FAULT, L_REAR,</td>
</tr>
<tr>
<td>0073</td>
<td>D2A1</td>
<td>75</td>
<td>SETB L_REAR,</td>
</tr>
<tr>
<td>0075</td>
<td>C296</td>
<td>76</td>
<td>CLR R_FRNT,</td>
</tr>
<tr>
<td>0077</td>
<td>208413</td>
<td>77</td>
<td>JB TO_FAULT, R_FRNT,</td>
</tr>
<tr>
<td>007A</td>
<td>D296</td>
<td>78</td>
<td>SETB R_FRNT,</td>
</tr>
<tr>
<td>007C</td>
<td>C2A0</td>
<td>79</td>
<td>CLR R_DASH,</td>
</tr>
<tr>
<td>007E</td>
<td>20840C</td>
<td>80</td>
<td>JB TO_FAULT, R_DASH,</td>
</tr>
<tr>
<td>0081</td>
<td>D2A0</td>
<td>81</td>
<td>SETB R_DASH, AND R_REAR</td>
</tr>
<tr>
<td>0083</td>
<td>C2A2</td>
<td>82</td>
<td>CLR R_REAR,</td>
</tr>
<tr>
<td>0085</td>
<td>208405</td>
<td>83</td>
<td>JB TO_FAULT,</td>
</tr>
<tr>
<td>0088</td>
<td>D2A2</td>
<td>84</td>
<td>SETB R_REAR,</td>
</tr>
<tr>
<td>008A</td>
<td>208402</td>
<td>85</td>
<td>JB TO_FAULT,</td>
</tr>
<tr>
<td>008D</td>
<td>D2A3</td>
<td>86</td>
<td>FAULT, CPL S_FAIL, ELECTRICAL FAILURE PROCESSING ROUTINE</td>
</tr>
<tr>
<td>99</td>
<td>+1</td>
<td>87</td>
<td>$EJECT</td>
</tr>
</tbody>
</table>

**Notes:**
- LOC: Location
- OBJ: Object Code
- LINE: Line Number
- SOURCE: Source Code
- ORG: Organization
- LJMP: Jump to Location
- MOV: Move
- PUSH: Push
- SETB: Set Bit
- CLR: Clear Bit
- JB: Jump if Bit
- SET: Set Bit
- CPL: Complement
- FAULT: Fault
- TO: To
- TOSERV: To Service
- RESET: Reset
100 CONTINUE WITH INTERRUPT PROCESSING:
101
102 1) COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON.
103
104 TOSERV: MOV C.SUB_DIV 1 , START WITH 50 PERCENT.
00BF A201
0091 B200
0093 7202
0095 B292
0097 92D1
105 ANL C.SUB_DIV 0 ; MASK DOWN TO 25 PERCENT.
0099 A293
009B 7291
009D B207
009F 9297
106 ORL C.SUB_DIV 2 ; BUILD BACK TO 62.5 PERCENT.
107 ANL C.PARK ; GATE WITH PARKING LIGHT SWITCH.
108 MOV DIM.C , AND SAVE IN TEMP. VARIABLE.
109
110 2) COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR
111
112 MOVC.L_TURN ; SET CARRY IF TURN
0099 A293
009B 7291
009D B207
009F 9297
113 ORL C.EMERG ; OR EMERGENCY SELECTED.
114 ANL C.LO_FRE < i ; IF SO, GATE IN 1 HZ SIGNAL
115 MOV L_DASH.C AND OUTPUT TO DASHBOARD.
116
117 3) COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.
118
119 MOVF.O.C ; SAVE FUNCTION SO FAR.
00A1 92D5
00A3 72D1
00A5 9295
120 ORL C.DIM ; ADD IN PARKING LIGHT FUNCTION
00A7 A290
00A9 B093
00AB 72D5
121 MOV L.FRNT.C AND OUTPUT TO TURN SIGNAL.
00AD 72D1
00AF 92A1
122
123 4) COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.
124
125 MOV C.BRAKE ; GATE BRAKE PEDAL SWITCH
00A7 A290
00A9 B093
00AB 72D5
126 ANLC.L_TURN ; WITH TURN LEVER
00AD 72D1
00AF 92A1
127 ORL C.FO ; INCLUDE TEMP. VARIABLE FROM DASH
128 ORL C.DIM ; AND PARKING LIGHT FUNCTION
129 MOV L_REAR.C AND OUTPUT TO TURN SIGNAL.
130
131 5) REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.
132
133 MOVC.R_TURN ; SET CARRY IF TURN
00B1 A294
00B3 7291
00B5 B207
00B7 92A0
00B9 92D5
134 ORL C.EMERG ; OR EMERGENCY SELECTED.
00BB 72D1
00BD 9296
135 ANLC.LO_FRE < i ; IF SO, GATE IN 1 HZ SIGNAL
136 MOV R_DASH.C AND OUTPUT TO DASHBOARD.
00BF A290
00C1 B094
00C3 72D5
137 MOVFO.C ; SAVE FUNCTION SO FAR.
00C5 72D1
00C7 92A2
138 ORL C.DIM ; ADD IN PARKING LIGHT FUNCTION
139 MOV R.FRNT.C AND OUTPUT TO TURN SIGNAL.
140 MOV C.BRAKE ; GATE BRAKE PEDAL SWITCH
00C9 D0D0
00CB 32
141 ANLC.R_TURN ; WITH TURN LEVER.
142 ORL C.FO ; INCLUDE TEMP. VARIABLE FROM DASH
00CF 92A2
143 ORL C.DIM ; AND PARKING LIGHT FUNCTION
144 MOV R_REAR.C AND OUTPUT TO TURN SIGNAL.
145
146 RESTORE STATUS REGISTER AND RETURN.
147
148 POP PSW ; RESTORE PSW
00C9 D0D0
00CB 32
149 RETI ; AND RETURN FROM INTERRUPT ROUTINE
150
151 END
## XREF SYMBOL TABLE LISTING

### NAME TYPE VALUE AND REFERENCES

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE AND REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRAKE</td>
<td>N BSEG</td>
<td>0090H 20# 125 140</td>
</tr>
<tr>
<td>DIM</td>
<td>N BSEG</td>
<td>00D1H 45# 108 120 128 138 143</td>
</tr>
<tr>
<td>EA</td>
<td>N BSEG</td>
<td>00AFH 6#</td>
</tr>
<tr>
<td>EMERG</td>
<td>N BSEG</td>
<td>0091H 21# 113 134</td>
</tr>
<tr>
<td>ETO</td>
<td>N BSEG</td>
<td>00A9H 63</td>
</tr>
<tr>
<td>FO</td>
<td>N BSEG</td>
<td>00D5H 119 127 137 142</td>
</tr>
<tr>
<td>FAULT</td>
<td>L CSEG</td>
<td>00BDH 75 78 81 84 87 90 97#</td>
</tr>
<tr>
<td>HI_FREQ</td>
<td>N BSEG</td>
<td>0000H 42#</td>
</tr>
<tr>
<td>INIT</td>
<td>L CSEG</td>
<td>0040H 50 58#</td>
</tr>
<tr>
<td>L_DASH</td>
<td>N BSEG</td>
<td>0097H 32# 77 79 115</td>
</tr>
<tr>
<td>L_FRNT</td>
<td>N BSEG</td>
<td>0095H 30# 74 76 121</td>
</tr>
<tr>
<td>L_REAR</td>
<td>N BSEG</td>
<td>00A1H 34# 80 82 129</td>
</tr>
<tr>
<td>L_TURN</td>
<td>N BSEG</td>
<td>0093H 23# 112 126</td>
</tr>
<tr>
<td>LD_FREQ</td>
<td>N BSEG</td>
<td>0007H 43# 114 135</td>
</tr>
<tr>
<td>P1</td>
<td>N DSEG</td>
<td>0090H 20 21 22 23 24 30 31 32 72</td>
</tr>
<tr>
<td>P2</td>
<td>N DSEG</td>
<td>00A0H 33 34 35 37 73</td>
</tr>
<tr>
<td>PARK</td>
<td>N BSEG</td>
<td>0092H 22# 107</td>
</tr>
<tr>
<td>PSW</td>
<td>N DSEG</td>
<td>00D0H 45 54 148</td>
</tr>
<tr>
<td>R_DASH</td>
<td>N BSEG</td>
<td>00A0H 33# 86 88 136</td>
</tr>
<tr>
<td>R_FRNT</td>
<td>N BSEG</td>
<td>0096H 31# 83 85 139</td>
</tr>
<tr>
<td>R_REAR</td>
<td>N BSEG</td>
<td>00A2H 35# 89 91 144</td>
</tr>
<tr>
<td>R_TURN</td>
<td>N BSEG</td>
<td>0094H 24# 133 141</td>
</tr>
<tr>
<td>S_FAIL</td>
<td>N BSEG</td>
<td>00A3H 37# 97</td>
</tr>
<tr>
<td>SUB_DIV</td>
<td>N DSEG</td>
<td>0020H 41# 42 43 62 69 70 104 105 106</td>
</tr>
<tr>
<td>TO</td>
<td>N BSEG</td>
<td>00B4H 75 78 81 84 87 90 96</td>
</tr>
<tr>
<td>TOServ</td>
<td>L CSEG</td>
<td>00BH 69 96 104#</td>
</tr>
<tr>
<td>THO</td>
<td>N DSEG</td>
<td>00BCH 53 59</td>
</tr>
<tr>
<td>TLO</td>
<td>N DSEG</td>
<td>00BAH 58</td>
</tr>
<tr>
<td>TMOD</td>
<td>N DSEG</td>
<td>0099H 60</td>
</tr>
<tr>
<td>TRO</td>
<td>N BSEG</td>
<td>00BCH 65</td>
</tr>
<tr>
<td>UPDATE</td>
<td>L CSEG</td>
<td>0044H 55 69#</td>
</tr>
</tbody>
</table>

---

**Assembly Complete, No Errors Found**
8051 Based CRT Terminal Controller

Michael A. Shafer
Microcontroller Applications
1.0 INTRODUCTION

This is the third application note that Intel has produced on CRT terminal controllers. The first Ap Note (ref. 1), written in 1977, used the 8080 as the CPU and required 41 packages including 11 LSI devices. In 1979, another application note (ref. 2) using the 8085 as the controller was produced and the chip count decreased to 20 with 11 LSI devices.

Advancing technology has integrated a complete system onto a single device that contains a CPU, program memory, data memory, serial communication, interrupt controller, and I/O. These “computer-on-a-chip” devices are known as microcontrollers. Intel’s MCS®-51 microcontroller was chosen for this application because of its highly integrated functions. This CRT terminal design uses 12 packages with only 4 LSI devices.

This application note has been divided into five general sections:
1) CRT Terminal Basics
2) 8051 Description
3) 8276 Description
4) Design Background
5) System Description

2.0 CRT TERMINAL BASICS

A terminal provides a means for humans to communicate with a computer. Terminals may be as simple as a LED display and a couple of push buttons, or it may be an elaborate graphics system that contains a full function keyboard with user programmable keys, color CRT and several processors controlling its functions. This application note describes a basic low cost terminal containing a black and white CRT display, full function keyboard and a serial interface.

2.1 CRT Description

A raster scan CRT displays its images by generating a series of lines (raster) across the face of the tube. The electron beam usually starts at the top left hand corner moves left to right, back to the left of the screen, moves down one row and continues on to the right. This is repeated until the lower right hand of the screen is reached. Then the beam returns to the top left hand corner and refreshes the screen. The beam forms a zigzag pattern as shown in Figure 2.1.0.

Two independent operating circuits control this movement across the screen. The horizontal oscillator controls the left to right motion of the beam while the vertical controls the top to bottom movement. The vertical oscillator also tells the beam when to return to the upper left hand corner or “home” position.

As the electron beam moves across the screen under the control of the horizontal oscillator, a third circuit controls the current entering the electron gun. By varying the current, the image may be made as bright or as dim as the user desires. This control is also used to turn the beam off or “blank the screen”.

When the beam reaches the right hand side of the screen, the beam is blanked so it does not appear on the screen as it returns to the left side. This “retrace” of the beam is at a much faster rate than it traveled across the screen to generate the image.

The time it takes to scan the whole screen and return to the home position is referred to as a “frame”. In the United States, commercial television broadcast uses a horizontal sweep frequency of 15.750Hz which calculates out to 63.5 microseconds per line. The frame time is equal to 16.67 milliseconds or 60Hz vertical sweep frequency.

Although this is the commercial standard, many CRT displays operate from 18KHz to 30KHz horizontal frequency. As the horizontal frequency increases, the number of lines per frame increases. This increase in lines or resolution is needed for graphic displays and on special text editors that display many more lines of text than the standard 24 or 25 character lines.

Since the United States operates on a 60Hz A.C. power line frequency, most CRT monitors use 60Hz as the vertical frequency. The use of 60Hz as the vertical frequency allows the magnetic and electrical variations that can modulate the electron beam to be synchronized with the display, thus they go unnoticed. If a frequency other than 60Hz is used, special shielding and power supply regul-
lating is usually required. Very few CRTs operate on a vertical frequency other than 60Hz due to the increase in the overall system cost.

The CRT controller must generate the pulses that define the horizontal and vertical timings. On most raster scan CRTs the horizontal frequency may vary as much as 500Hz without any noticeable effect on the quality of the display. This variation can change the number of horizontal lines from 256 to 270 per frame.

The CRT controller must also shift out the information to be displayed serially to the circuit that controls the electron beam’s intensity as it scans across the screen. The circuits that control the timing associated with the shifting of the information are known as the dot clock and the character clock. The character clock frequency is equal to the dot clock frequency divided by the number of dots it takes to form a character in the horizontal axis. The dot clock frequency is calculated by the following equation:

\[
\text{Dot Clock (Hz)} = (N + R) \times D \times L \times F
\]

where

- \(N\) is the number of displayed characters per row,
- \(R\) is the number of character times for the retrace,
- \(D\) is the number of dots per character in the horizontal axis,
- \(L\) is the number of horizontal lines per frame,
- \(F\) is the frame rate in Hz.

In this design \(N = 80\), \(R = 20\), \(D = 7\), \(L = 270\), and \(F = 60Hz\). Plugging in the numbers results in a dot clock frequency of 11.34MHz.

The retrace number may vary on each design because it is used to set the left and right hand margins on the CRT. The number of dots per character is chosen by the designer to meet the system needs. In this design, a 5 x 7 dot matrix and 2 blank dots between each character (see Figure 2.1.1) makes \(D\) equal to \(5 + 2 = 7\).

The following equation can be used to figure the number of lines per frame:

\[
L = (H \times Z) + V
\]

where

- \(H\) is the number of horizontal lines per character,
- \(Z\) is the number of character lines per frame,
- \(V\) is the number of horizontal line times during the vertical retrace.

In this design \(H\) is equal to the 7 horizontal dots per character plus 3 blank dots between each row which adds up to 10. Also 25 lines of characters are displayed, so \(Z = 25\). The vertical retrace time is variable to set the top and bottom margins on the CRT and in this design is equal to 20. Plugging in the numbers gives \(L = 270\) lines per frame.

### 2.2 Keyboard

A keyboard is the common way a human enters commands and data to a computer. A keyboard consists of a matrix of switches that are scanned every couple of milliseconds by a keyboard controller to determine if one of the keys has been pressed. Since the keyboard is made up of mechanical switches that tend to bounce or "make and break" contact everytime they are pressed, debouncing of the switches must also be a function of the keyboard controller. There are dedicated keyboard controllers available that do everything from scanning the keyboard, debouncing the keys, decoding the ASCII code for that key closure to flagging the CPU that a valid key has been depressed. The keyboard controller may present the information to the CPU in parallel form or in a serial data stream.

This Application Note integrates the function of the keyboard controller into the 8051 which is also the terminal controller. Provisions have been made to interface the 8051 to a keyboard that uses a dedicated keyboard controller. The 8051 can accept data from the keyboard controller in either parallel or serial format.

### 2.3 Serial Communications

Communication between a host computer and the CRT terminal can be in either parallel or serial data format. Parallel data transmission is needed in high end graphic terminals where great amounts of information must be transferred.

One can rarely type faster than 120 words per minute, which corresponds to 12 characters per second or 1 character per 83 milliseconds. The utilization of a parallel port cannot justify the cost associated with the drivers and the amount of wire needed to perform this transmission. Full duplex serial data transmission requires 3 wires and two
drivers to implement the communication channel between the host computer and the terminal. The data rate can be as high as 19200 BAUD in the asynchronous serial format. BAUD rate is the number of bits per second received or transmitted. In the asynchronous serial format, 10 bits of information is required to transmit one character. One character per 500 microseconds or 1,920 characters per second would then be transmitted using 19.2 KBAUD.

This application note uses the 8051 serial port configured for full duplex asynchronous serial data transmission. The software for the 8051 has been written to support variable BAUD rates from 150 BAUD up to 9.6 KBAUD.

3.0 8051 DESCRIPTION

The 8051 is a single chip high-performance microcontroller. A block diagram is shown in figure 3.0.0. The 8051 combines CPU; Boolean processor; 4K × 8 ROM: 128 × 8 RAM; 32 I/O lines; two 16-bit timer/ event counters; a five-source, two-priority-level, nested interrupt structure; serial I/O port for either multiprocessor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits.

3.1 CPU

Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, branches may use either an offset that is relative to the program counter which takes two bytes or a direct 16-bit address which takes three bytes to perform. As a result, 64 instructions operate in one machine cycle, 45 in two machine cycles, and the multiply and divide instruction execute in 4 machine cycles.

The 8051 has five addressing modes for source operands. The Boolean Processor can be thought of as a separate one-bit CPU. It has its own accumulator (the carry bit), instruction set for data moves, logic, and control transfer, and its own bit addressable RAM and I/O. The bit-manipulating instructions provide optimum code and speed efficiency for handling on chip peripherals. The
Boolean processor also provides a straightforward means of converting logic equations directly into software. Complex combinational logic functions can be resolved without extensive data movement, byte masking, and test-and-branch trees.

3.2 On-Chip RAM

The CPU manipulates operands in four memory spaces. These are the 64K-byte Program Memory, 64K-byte External Data Memory, 128-byte Internal Data Memory, and 128-byte Special Function Registers (SFRs). Four Register Banks (each with 8 registers), 128 addressable bits, and the Stack reside in the internal Data RAM. The Stack size is limited only by the available Internal Data RAM and its location is determined by the 8-bit Stack Pointer. All registers except for the Program Counter and the four 8-Register Banks reside in the SFR address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, and registers for the interrupt system, timers, and serial channel.

Registers in the four 8-Register Banks can be addressed by Register, Direct, or Register-Indirect Addressing modes. The 128 bytes of internal Data Memory can be addressed by Direct or Register-Indirect modes while the SFRs are only addressed directly.

3.3 I/O Ports

The 8051 has instructions that can treat the 32 I/O lines as 32 individually addressable bits or as 4 parallel 8-bit ports addressable as Ports 0, 1, 2, and 3.

Resetting the 8051 writes a logical 1 to each pin on port 0 which places the output drivers into a high-impedance mode. Writing a logical 0 to a pin forces the pin to ground and sinks current. Re-writing the pin high will place the pin in either an open drain output or high-impedance input mode.

Ports 1, 2, and 3 are known as quasi-bidirectional I/O pins. Resetting the device writes a logical one to each pin. Writing a logical 0 to the pin will force the pin to ground and sink current. Re-writing the pin high will place the pin in an output mode with a weak depletion pullup FET or in the input mode. The weak pullup FET is easily overcome by a TTL output.

Ports 0 and 2 can also be used for off-chip peripheral expansion. Port 0 provides a multiplexed low-order address and data bus while Port 2 contains the high-order address when using external Program Memory or more than 256 byte external Data Memory.

Port 3 pins can also be used to provide external interrupt request inputs, event counter inputs, the serial port TXD and RXD pins and to generate control signals used for writing and reading external peripherals.

3.4 Interrupt System

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. A five-source, two-level, nested interrupt system ties the real time events to the normal program execution.

The 8051 has two external interrupt sources, one interrupt from each of the two timer/counters, and an interrupt from the serial port. Each interrupt vector the program execution to its own unique memory location for servicing the interrupt. In addition, each of the five sources can be individually enabled or disabled as well as assigned to one of the two interrupt priority levels available on the 8051.

Up to two additional external interrupts can be created by configuring a timer/counter to the event counter mode. In this mode the timer/counter increments on command by either the T0 or T1 pin. An interrupt is generated when the timer/counter overflows. Thus if the timer/counter is loaded with the maximum count, the next high-to-low transition of the event counter input will cause an interrupt to be generated.

3.5 Serial Port

The 8051's serial port is useful for linking peripheral devices as well as multiple 8051s through standard asynchronous protocols with full duplex operation. The serial port also has a synchronous mode for expansion of I/O lines using shift registers. This hardware serial port saves ROM code and permits a much higher transmission rate than could be achieved through software. The processor merely needs to read or write the serial buffer in response to an interrupt. The receiver is double buffered to eliminate the possibility of overrun if the processor failed to read the buffer before the beginning of the next frame.

The full duplex asynchronous serial port provides the means of communication with standard UART devices such as CRT terminals and printers.

The reader should refer to the microcontroller handbook for a complete discussion of the 8051 and its various modes of operation.

4.0 8276 DESCRIPTION

The 8276's block diagram and pin configuration are shown in Figure 4.0.0. The following sections describe the general capabilities of the 8276.
4.1 CRT Display Refreshing

The 8276, having been programmed by the system designer for a specific screen format, generates a series of Buffer Ready signals. A row of characters is then transferred by the system controller from the display memory to the 8276’s row buffers. The row buffers are filled by deselecting the 8276 CS and asserting the BS and WR signals. The 8276 presents the character codes to an external character generator ROM by using outputs CC0–CC6. The parallel data from the outputs of the character generator is converted to serial information that is clocked by external dot timing logic into the video input of the CRT.

The character rows are displayed on the CRT one line at a time. Line count outputs LC0–LC3 select the current line information from the character generator ROM. The display process is illustrated in Figure 4.1.0. This process is repeated for each display character row. At the beginning of the last display row the 8276 generates an interrupt request by raising its INT output line. The interrupt request is used by the 8051 system controller to reinitialize its load buffer pointers for the next display refresh cycle.

Proper CRT refreshing requires that certain 8276 parameters be programmed at system initialization time. The 8276 has two types of internal registers; the write only Command (CREG) and Parameter (PREG) Registers, and the read only Status Register (SREG). The 8276 expects to receive a command followed by 0 to 4 parameter bytes depending on the command. A summary of the 8276’s instruction set is shown in Figure 4.1.1. To access the registers, CS must be asserted along with WR or RD. The status of the C/P pin determines whether the command or parameter registers are selected.

The 8276 allows the designer flexibility in the display format. The display may be from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per character row. In addition, four cursor formats are available; blinking, non-blinking, underline, and reverse video. The cursor position is programmable to anywhere on the screen via the Load Cursor command.
4.2 CRT Timing

The 8276 provides two timing outputs for controlling the CRT. The Horizontal Retrace Timing and Control (HRTC) and Vertical Retrace Timing and Control (VRTC) signals are used for synchronizing the CRT horizontal and vertical oscillators. A third output, VSP (Video Suppress), provides a signal to the dot timing logic to blank the video signal during the horizontal and vertical retraces. LTEN (Light Enable) is used to provide the ability to force the video output high regardless of the state of the VSP signal. This feature is used to place the cursor on the screen and to control attribute functions.

RVV (Reverse Video) output, if enabled, will cause the system to invert its video output. The fifth timing signal output, HLGT (highlight) allows the flexibility to increase the CRT beam intensity to a greater than normal level.
### Table

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>NO. OF PARAMETER BYTES</th>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>4</td>
<td>Display format parameters required</td>
</tr>
<tr>
<td>START DISPLAY</td>
<td>0</td>
<td>DMA operation parameters included in command</td>
</tr>
<tr>
<td>STOP DISPLAY</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>RED LIGHT PEN</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>LOAD CURSOR</td>
<td>2</td>
<td>Cursor X, Y position parameters required</td>
</tr>
<tr>
<td>ENABLE INTERRUPT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DISABLE INTERRUPT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>PRESET COUNTERS</td>
<td>0</td>
<td>Clears all internal counters</td>
</tr>
</tbody>
</table>

4) **End Of Screen-Stop Buffer Loading Code** — Causes the Buffer Ready control logic to stop requesting buffer transfers until the end of the frame is reached. It affects the display the same way as the End of Screen Code.

### 4.3 Special Functions

#### 4.3.1 Special Codes

The 8276 recognizes four special codes that may be used to reduce memory, software, or system controller overhead. These characters are placed within the display memory by the system controller. The 8276 performs certain tasks when these codes are received in its row buffer memory.

1) **End of Row Code** — Activates VSP. VSP remains active until the end of the line is reached. While VSP is active the screen is blanked.

2) **End Of Row-Stop Buffer Loading Code** — Causes the Buffer Ready control logic to stop requesting buffer transfers for the rest of the row. It affects the display the same as End of Row Code.

3) **End Of Screen Code** — Activates VSP. VSP remains active until the end of the frame is reached.

4) **End Of Screen-Stop Buffer Loading Code** — Causes the Buffer Ready control logic to stop requesting buffer transfers until the end of the frame is reached. It affects the display the same way as the End of Screen Code.

#### 4.3.4 Programmable Buffer Loading Control

The 8276 can be programmed to request 1, 2, 4, or 8 characters per Buffer load. The interval between loads is also programmable. This allows the designer the flexibility to tailor the buffer transfer overhead to fit the system needs.

Each scan line requires 63.5 microseconds. A character line consists of 10 scan lines and takes 635 microseconds to form. The 8276 row buffer must be filled within the 635 microseconds or an under run condition will occur within the 8276 causing the screen to be blanked until the next vertical retrace. This blanking will be seen as a flicker in the display.

### 5.0 DESIGN BACKGROUND

A fully functional, microcontroller-based CRT terminal was designed and constructed using the 8051 and the 8276. The terminal has many of the functions that are found in commercially available low cost terminals. Sophisticated features such as programmable keys can be added easily with modest amounts of software.

The 8051's functions in this application note include: up to 9.6K BAUD full duplex serial transmission; decoding special messages sent from the host computer; scanning, debouncing, and decoding a full function keyboard; writing to the 8276 row buffer from the display RAM without the need for a DMA controller; and scrolling the display.

The 8276 CRT controller's functions include: presenting the data to the character generator; providing the timing signals needed for horizontal and vertical retrace; and providing blanking and video information.

#### 5.1 Design Philosophy

Since the device count relates to costs, size, and reliability of a system, arriving at a minimum device count without degrading the performance was a driving force for this application note. LSI devices were used where possible to maintain a low chip count and to make the design cycle as short as possible.

PL/M-51 was chosen to generate the majority of the software for this application because it models the human thought process more closely than assembly language. Consequently it is easier and faster to write programs using PL/M-51 and the code is more likely to be correct because less chance exists to introduce errors.
PL/M-51 programs are easier to read and follow than assembly language programs, and thus are easier to modify and customize to the end user's application. PL/M-51 also offers lower development and maintenance costs than assembly language programming.

PL/M-51 does have a few drawbacks. It is not as efficient in code generation relative to assembly language and thus may also run slower.

This application note uses the 8051's interrupts to control the servicing of the various peripherals. The speed of the main program is less critical if interrupts are used. In the last two application notes on terminal controllers, a criterion of the system was the time required for receiving an incoming serial byte, decoding it, performing the function requested, scanning the keyboard, debouncing the keys, and transmitting the decoded ASCII code must be less than the vertical refresh time. Using the 8051 and its interrupts makes this time constraint irrelevant.

5.2 System Target Specifications

The design specifications for the CRT terminal design is as follows:

Display Format
- 80 characters/display row
- 25 display lines

Character Format
- 5 × 7 character contained within a 7 × 10 frame
- First and seventh columns blanked
- Ninth line curser position
- Programmable delay blinking underline curser

Control Characters Recognized
- Backspace
- Linefeed
- Carriage Return
- Form Feed

Escape Sequences Recognized
- ESC A, Curser up
- ESC B, Curser down
- ESC C, Curser right
- ESC D, Curser left
- ESC E, Clear screen
- ESC F, Move addressable curser
- ESC H, Home curser
- ESC J, Erase from curser to the end the screen
- ESC K, Erase the current line

Characters Displayed
- 96 ASCII Alphanumeric Characters

Characters Transmitted
- 96 ASCII Alphanumeric Characters
- ASCII Control Character Set
- ASCII Escape Sequence Set
- Auto Repeat

Display Memory
- 2K × 8 static RAM

Data Rate
- Variable rate from 150 to 9600 BAUD

CRT Monitor
- Ball Bros TV-12, 12MHZ Black and White

Keyboard
- Any standard undecoded keyboard (2 key lock-out)
- Any standard decoded keyboard with output enable pin
- Any standard decoded serial keyboard up to 150 BAUD

Scrolling Capability
Compatible With Wordstar

6.0 SYSTEM DESCRIPTION

A block diagram of the CRT terminal is shown in figure 6.0.0. The diagram shows only the essential system features. A detailed schematic of the CRT terminal is contained in the Appendix 7.1.

The "brains" of the CRT terminal is the 8051 microcontroller. The 8276 is the CRT controller in the system, and a 2716 EPROM is used as the character generator. To handle the high speed portion of the CRT, the 8276 is surrounded by a handful of TTL devices. A 2K × 8 static RAM was used as the display memory.

Following the system reset, the 8276 is initialized for curser type, number of characters per line, number of lines, and character size. The display RAM is initialized to all "spaces" (ASCII 20H). The 8051 then writes the "start display" command to the 8276. The local/line input is sampled to determine the terminal mode. If the terminal is on-line, the BAUD rate switches are read and the serial port is set up for full duplex UART mode. The processor then is put into a loop waiting to service the serial port fifo or the 8276.

The serial port is programmed to have the highest priority interrupt. If the serial port generates an interrupt, the processor reads the buffer, puts the character in a generated fifo that resides in the 8051's internal RAM, increments the fifo pointer, sets the serial interrupt flag and returns.
The main program determines if it is a displayable character, a Control word or an ESC sequence and either puts the character in the display buffer or executes the appropriate command sent from the host computer.

If the 8276 needs servicing, the 8051 fills the row buffer for the CRT display's next line. If the 8276 generates a vertical retrace interrupt, the buffer pointers are reloaded with the display memory location that corresponds to the first character of the first display line on the CRT. The vertical retrace also signals the processor to read the keyboard for a key closure.

### 6.1 Hardware Description

The following section describes the unique characteristics of this design.

#### 6.1.1 Peripheral Address Map

The display RAM, 8276 registers, and the 8276 row buffers are memory mapped into the external data RAM address area. The addresses are as follows:

- **Read and Write External Display RAM** — Address 1000H to 17CFH
- **Write to 8276 row buffers from Display RAM** — Address 1800H to 1FCFH
- **Write to 8276 Command Register (CREG)** — Address 0001H
- **Write to 8276 Parameter Register (PREG)** — Address 0000H
- **Read from 8276 Status Register (SREG)** — Address 0001H

Three general cases can be explored; reading and writing the display RAM, writing to the 8276 row buffers, and reading and writing the 8276's control registers.

As mentioned previously the 8051 fills the 8276 row buffer without the need of a DMA controller. This is accomplished by using a Quad 2-input multiplexor (Figure 6.1.0) as the transfer logic shown in the block diagram. The address line, P2.3, is used to select either of the two inputs. When the address line is low the RD and WR lines perform their normal functions, that is read and write the...
8276 or the external display RAM depending on the states of their respective chip selects. If the address line is high, the 8051 RD line is transformed into BS and WR signals for the 8276. While holding the address line high, the 8051 executes an external data move (MOVX) from the display RAM to the accumulator which causes the display RAM to output the addressed byte onto the data bus. Since the multiplexer turns the same 8051 RD pulses into BS and WR pulses to the 8276, the data bus is thus read into the 8276 as a Buffer transfer. This scheme allows 80 characters to be transferred from the display RAM into the 8276 within the required character line time of 635 microseconds. The 8051 easily meets this requirement by accomplishing the task within 350 microseconds.

6.1.2 Scanning The Keyboard

Throughout this project, provision have been made to make the overall system flexible. The software has been written for various keyboards and the user simply needs to link different program modules together to suit their needs.

6.1.2.1 Undecoded Keyboard

Incorporating an undecoded keyboard controller into the other functions of the 8051 shows the flexibility and overall CPU power that is available. The keyboard in this case is a full function, non-buffered 8 x 8 matrix of switches for a total of 64 possible keys. The 8 send lines are connected to a 3-to-8 open-collector decoder as shown in Figure 6.1.1. Three high order address lines from the 8051 are the decoder inputs. The enabling of the decoder is accomplished through the use of the PSEN signal from the 8051 which makes the architecture of the separate address space for the program memory and the external data RAM work for us to eliminate the need to decode addresses externally. The move code (MOVC) instruction allows each scan line of the keyboard to be read with one instruction.

The keyboard is read by bringing one of the eight scan lines low sequentially while reading the return lines which are pulled high by an external resistor. If a switch is
closed, the data bus line is connected through the switch to the low output of the decoder and one of the data bus lines will be read as a 0. By knowing which scan line detected a key closure and which data bus line was low, the ASCII code for that key can easily be looked up in a matrix of constants. PL/M-51 has the ability to handle arrays and structured arrays, which makes the decoding of the keyboard a trivial task.

Since the Shift, Cap Lock, and Control keys may change the ASCII code for a particular key closure, it is essential to know the status of these pins while decoding the keyboard. The Shift, Cap Lock, and Control keys are therefore not scanned but are connected to the 8051 port pins where they can be tested for closure directly.

The 8 receive lines are connected to the data bus through germanium diodes which chosen for their low forward voltage drop. The diodes keep the keyboard from interfering with the data bus during the times the keyboard is not being read. The circuit consisting of the 3-to-8 decoder and the diodes also offers some protection to the 8051 from possible Electrostatic Discharge (ESD) damage that could be transmitted through the keyboard.

6.1.2.2 Decoded Keyboard

A decoded keyboard can easily be connected to the system as shown in Figure 6.1.2. Reading the keyboard can be evoked either by interrupts or by software polling.

The software to periodically read a decoded keyboard was not written for this application note but can be accomplished with one or two PL/M-51 statements in the READER routine.

A much more interesting approach would be to have the servicing of the keyboard be interrupt driven. An additional external interrupt is created by configuring timer/counter 0 into an event counter. The event counter is initialized with the maximum count. The keyboard controller would inform the 8051 that a valid key has been depressed by pulling the input pin T0 low. This would overflow the event counter, thus causing an interrupt. The interrupt routine would simply use a MOV C (PSEN is connected to the output enable pin of the keyboard controller) to read the contents of the keyboard controller onto the data bus, reinitialize the counter to the maximum count and return from the interrupt.

6.1.2.3 Serial Decoded Keyboard

The use of detachable keyboards has become popular among the manufacturers of keyboards and personal computers. This terminal has provisions to use such a keyboard.

The keyboard controller would scan the keyboard, debounce the key and send back the ASCII code for that key closure. The message would be in an asynchronous serial format.

The flowchart for a software serial port is shown in Figure 6.1.3. An additional external interrupt is created as discussed for the decoded keyboard but the use in this case would be to detect a start bit. Once the beginning of the start bit has been detected, the timer/counter 0 is configured to become a timer. The timer is initialized to cause an interrupt one-half bit time after the beginning of the start bit. This is to validate the start bit. Once the start bit is validated, the timer is initialized with a value to cause an interrupt one bit time later to read the first data bit. This process of interrupting to read a data bit is repeated until all eight data bits have been received. After all 8 data bits are read, the software serial port is read once more to detect if a stop bit is present. If the stop bit is not present, an error flag is set, all pointers and flags are reset to their initial values, and the timer/counter is reconfigured to an event counter to detect the next start bit. If the stop bit is present, a valid flag is set and the flags and counter are reset as previously discussed.

![Figure 6.1.2 Using A Decoded Keyboard](image-url)
6.1.4 System Timings

The requirements for the BALL BROTHERS. TV-12 monitor's operation is shown in table 6.1.0. From the monitor's parameters, the 8276 specifications and the system target specifications the system timing is easily calculated.

The 8276 allows the vertical retrace to be only an integer multiple of the horizontal character lines. Twenty-five display lines and a character frame of $7 \times 10$ are required from the target specification which will require 250 horizontal lines. If the horizontal frequency is to be within 500 Hz of 15,750 Hz, we must choose either one or two character line times for horizontal retrace. To allow for a little more margin at the top and bottom of the screen, two character line times was chosen for the vertical retrace. This choice yields $250 + 20 = 270$ total character lines per frame. Assuming 60 Hz vertical retrace frequency:

$$60 \text{ Hz} \times 270 = 16,200 \text{ Hz}$$

and

$$\frac{1}{16,200 \text{ Hz}} \times 20 \text{ horizontal sync times} = 1.2345 \text{ milliseconds}$$

---

**Figure 6.1.3 Flowchart for the Software Serial Port**

---

10-77
Table 6.1.0 CRT Monitor's Operational Requirements

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical Blanking Time (VRTC)</td>
<td>800 ( \mu \text{sec} ) nominal</td>
</tr>
<tr>
<td>Vertical Drive Pulsewidth</td>
<td>300 ( \mu \text{sec} ) ( \leq ) PW ( \leq ) 1.4 ms</td>
</tr>
<tr>
<td>Horizontal Blanking Time (HRTC)</td>
<td>11 ( \mu \text{sec} ) nominal</td>
</tr>
<tr>
<td>Horizontal Drive Pulsewidth</td>
<td>25 ( \mu \text{sec} ) ( \leq ) PW ( \leq ) 30 ( \mu \text{sec} )</td>
</tr>
<tr>
<td>Horizontal Repetition Rate</td>
<td>15,750 ( \pm ) 500 pps</td>
</tr>
</tbody>
</table>

The 1.2345 milliseconds of retrace time meets the nominal VRTC and vertical drive pulse width time of .3mSec to 1.4mSec for the Ball monitor.

The next parameter to find is the horizontal retrace time which is wholly dependent on the monitor used. Usually it lies between 15 and 30 percent of the total horizontal line time.

Since most designs display a fixed number of characters per line it is useful to express the horizontal retrace time as a given number of character times. In this design, 80 characters are displayed, and it was experimentally found that 20 character times for the horizontal retrace gave the best results. It should be noted if too much time was given for retrace, there would be less time to display the characters and the display would not fill out the screen. Conversely, if not enough time is given for retrace, the characters would seem to run off the screen.

One hundred character times per complete horizontal line means that each character needs:

\[
\frac{1}{16,200 \text{ Hz}} / 100 \text{ character times} = 617.3 \text{ nanoseconds}
\]

If we multiply the 20 character times needed to retrace by 617.3 nanoseconds needed for each character, we find 12,345 microseconds are allocated for retrace. This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, a 74LS123 one-shot was used to extend the horizontal drive pulse width.

The dot clock frequency is easy to calculate now that we know the horizontal frequency. Since each character is formed by seven dots in the horizontal axis, the dot clock period would be the character clock (617.3 nanoseconds) divided by the 7 which is equal to 11.34 MHz. The basic dot timing and CRT timing are shown in the Appendix.

6.2 Software Description

6.2.1 Software Overview

The software for this application was written in a "foreground-background" format. The background programs are all interrupt driven and are written in assembly language due to time constraints. The foreground programs are the most part written in PL/M-51 to ease the programming effort. A number of subroutines are written in assembly language due to time constraints during execution. Subroutines such as clearing display lines, clearing the screen, and scanning the keyboard require a great deal of 16 bit adds and compares and would execute much slower and would require more code space if written in PL/M-51. The background and foreground programs talk to each other through a set of flags. For example, the PL/M-51 foreground program tests "SERIAL$INT" to determine if a serial port interrupt had occurred and a character is waiting to be processed.

6.2.2 The Background Program

Two interrupt driven routines, VERT and BUFFER, (see Fig. 6.2.0) request service every 16.67 milliseconds and 617 microseconds respectively. VERT's request comes during the last character row of the display screen. This routine resets the buffer pointers to the first CRT display line in the display memory. VERT is also used as a time base for the foreground program. VERT sets the flag, SCAN, to tell the foreground program (PL/M-51) that it is time to scan the Keyboard. VERT also increments a counter used for the delay between transmitting characters in the AUTO$REPEAT routine.

The BUFFER routine is executed once per character row. BUFFER uses the multiplexor discussed earlier to fill the 8276's row buffer by executing 80 external data moves and incrementing the Data Pointer between each move.
The MOVX reads the display RAM and writes the character into the row buffer during the same instruction.

SERBUF is an interrupt driven routine that is executed each time a character is received or transmitted through the on-chip serial port. The routine first checks if the interrupt was caused by the transmit side of the serial port, signaling that the transmitter is ready to accept another character. If the transmitter caused the interrupt, the flag “TRANSMIT$INT” is set which is checked by the foreground program before putting a character in the buffer for transmission.

If the receiver caused the interrupt, the input buffer on the serial port is read and fed into the fifo that has been manufactured in the internal RAM and increments the fifo pointer “FIFO.” The flag “SERIAL$INT” is then set, telling the foreground program that there is a character in the fifo to be processed. If the read character is an ESC character, the flag “ESCSEQ” is set to tell the foreground program that an escape sequence is in the process of being received.

6.2.3 The Foreground Program

The foreground program is documented in the Appendix. The foreground program starts off by initializing the 8276 as discussed earlier. After all variables and flags are initialized, the processor is put into a loop waiting for either VERT to set SCAN so the program can scan the keyboard, or for the serial port to set SERIAL$INT so the program can process the incoming character.

The vertical retrace is used to time the delay between keyboard scans. When VERT gets set, the assembly language routine READER is called. READER scans the keyboard, writing each scan into RAM to be processed later. READER controls two flags, KEY0 and SAME. KEY0 is set when all 8 scans determine that no key is pressed. SAME is set when the same key that was pressed last time the keyboard was read is still pressed.

After READER returns execution to the main program, the flags are tested. If the KEY0 flag is set the main program goes back to the loop waiting for the vertical retrace or a serial port interrupt to occur. If the SAME flag is set the main program knows that the closed key has been debounced and decoded so it sends the already known ASCII code to the AUTO$REPEAT routine which determines if that character should be transmitted or not.

If KEY0 and SAME are not set, signifying that a key is pressed but it is not the same key as before, the foreground program determines if the results from the scan are valid. First all eight scans are checked to see if only one key was closed. If only one key is closed, the ASCII code is determined, modified if necessary by the Shift, Cap Lock, or Control keys. The NEWSKEY and VALID flags are then set. The next time READER is called, if the same key is still pressed, the SAME flag will be set, causing the AUTO$REPEAT subroutine to be called as just discussed. Since the keyboard is read during the vertical retrace, 16.67 milliseconds has elapsed between the detection of the pressed key and revalidating that the key is still pressed before transmitting it, thus effectively debouncing the key.

The AUTO$REPEAT routine is written to transmit any key that the NEWSKEY flag is set for. The counter that is incremented each time the vertical refresh interrupt is serviced causes a programmable delay between the first transmission and subsequent auto repeat transmission. Once the NEWSKEY character is sent, the counter is initialized. Each time the AUTO$REPEAT routine is called, the counter is checked. Only when the counter overflows will the next character be transmitted. After the initial delay, a character will be transmitted every other time the routine is called as long as the key remains pressed.

6.2.3.1 Handling Incoming Serial Data

One of the criteria for this application note was to make the software less time dependent. By creating a fifo to store incoming characters until the 8051 has time to pro-
cess them, software timing becomes less critical. This application note uses up to 8 levels of the fifo at 9.2KBAUD, and 1 level at 4.8KBAUD and lower. As discussed earlier, the interrupt service routine for the serial port uses the fifo to store incoming data, increments the fifo pointer, "FIFO", and sets SERIAL$INT to tell the main program that the fifo needs servicing. Once the main program detects that SERIAL$INT is set the routine DECIPHER is executed.

DECIPHER has three separate blocks: a block for decoding displayable characters, a block for processing Escape sequences, and a block for processing Control codes. Each block works on the fifo independently. Before exiting a block, the contents of the fifo are shifted up by the amount of characters that were processed in that particular block. The shifting of the characters insures that the beginning of the fifo contains the next character to be processed. FIFO is then decremented by the number of characters processed.

Let’s look at this process more closely. Figure 6.2.1-A shows a representation of a fifo containing 5 characters. The first three characters in the fifo contain displayable characters, A, B, and C respectively with the last two characters being an ESC sequence for moving the curser up one line (ESC A) and FIFO points to the next available location to be filled by the serial port interrupt routine, in this case, 5.

When DECIPHER is executed, the first block begins looking at the first character of the fifo for a displayable character. If the character is displayable, it is placed into the display RAM and the software pointer “TOP” that points to the character that is being processed is incremented to the next character. The character is then looked at to see if it too is displayable and if it is, it’s placed in the display RAM. The process of checking for displayable characters is continued until either the fifo is empty or a non-displayable character is detected. In our example, three characters are placed into the display RAM before a non-displayable character is detected. At this point the fifo looks like figure 6.2.1-B.

Before entering the next block, the remaining contents of the fifo between TOP, that is now pointing to IBH and (FIFO-1) are moved up in the fifo by the amount of characters processed, in this example three. TOP is reset to 0 and FIFO is decremented by 3. The serial port interrupt is inhibited during the time the contents of the fifo and the pointers are being manipulated. The fifo now looks like figure 6.2.1-C.

The execution is now passed to the next block that processes ESC sequences. The first location of the fifo is examined to see if it is an ESC character (1BH). If not, the execution is passed to the next block of DECIPHER that processes Control codes. In this case the fifo does contain an ESC code. The flag ESC$SEQ is checked to see if the 8051 is in the process of receiving an ESC sequence thus signifying that the next byte of the sequence has not been received yet. If the ESC$SEQ is not set, the next character in the fifo is checked for a valid escape code and the proper subroutine is then called. The fifo contents are then shifted as discussed for the previous block. Due to the length of time that is needed to execute an ESC code sequence or a Control code, only one ESC code and/or Control code can be processed each time DECIPHER is executed.

If at the end of the DECIPHER routine, FIFO contains a 0, the flag SER$INT is reset. If SER$INT remains set, DECIPHER will be executed immediately after returning to the main program if SCAN had not been set during the execution of the DECIPHER routine, otherwise DECIPHER will be called after the keyboard is read.

6.2.4 Memory Pointers and Scrolling

The curser always points to the next location in display memory to be filled. Each time a character is placed in the display memory, the curser position needs to be tested to determine if the curser should be incremented to the beginning of the next line of the display or simply moved to the next position on the current display line. The curser position pointers are then updated in both the 8276 and the internal registers in the 8051.
When the 2000th character is entered into the display memory, a full display page has been reached signaling the need for the display to scroll. The memory pointer that points to the display memory that contains the first character of the first display line, LINE0, prior to scrolling contains 1800H which is the starting address of the display memory. Each scrolling operation adds 80 (50H) to LINE0 which will now point to the following row in memory as shown in figure 6.2.2-B. LINE0 is used during the vertical refresh routine to re-initialize the pointers associated with filling the 8276 row buffers.

The display memory locations that were the first line of the CRT display now becomes the last line of the CRT display. Incoming characters are now entered into the display memory starting with 1800H, which is now the first character of the last line of the display screen.
6.2.5 **Software Timing**

The use of interrupts to tie the operation of the foreground program to the real-time events of the background program has made the software timing non-critical for this system.

6.3 **System Operation**

Following the system reset, the 8051 initializes all on-chip peripherals along with the 8276 and display ram. After initialization, the processor waits until the fifo has a character to process or is flagged that it is time to scan the keyboard. This foreground program is interrupted once every 617 microseconds to service the 8276 row buffers. The 8051 is also interrupted each 16.67 milliseconds to re-initialize LINE0 and to flag the foreground program to read the keyboard.

As discussed earlier, a special technique of rapidly moving the contents of the display RAM to the 8276 row buffers without the need of a DMA device was employed. The characters are then synchronously transferred to the character generator via CC0-CC6 and LC0-LC2 which are used to display one line at a time. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line is selected. This process continues until the last line of the character is transferred.

The dot timing logic latches the output of the character ROM in a parallel in, serial out synchronous shift register. The shift register's output constitutes the video information to the CRT.
Appendix 7.1  CRT Schematics
Appendix 7.2  Dot Timing

![Diagram of Dot Timing](image)

- **Character Counter State Dot Clock**
- **74S163 Counter Outputs QC Clock Character Clock**
- **Character Clock to 8276**
- **8276 Character Output (CCD-CC6)**

**First Character**
- **First Character Video Out**

**Second Character**
- **Second Character Video Out**

**Third Character**

---

10-86
Appendix 7.3 CRT System Timing
Appendix 7.4  Escape/Control/Display Character Summary

<table>
<thead>
<tr>
<th>BIT</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>NUL</td>
<td>@</td>
<td>DLE</td>
<td>P</td>
<td>SP</td>
<td>#</td>
<td>@</td>
<td>P</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>SOH</td>
<td>DC1</td>
<td>Q</td>
<td>I</td>
<td>A</td>
<td>Q</td>
<td>A</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>STX</td>
<td>DC2</td>
<td>R</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>B</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>ETX</td>
<td>DC3</td>
<td>S</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>C</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>EOT</td>
<td>DC4</td>
<td>T</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>D</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>ENQ</td>
<td>NAK</td>
<td>U</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>E</td>
<td>U</td>
<td></td>
<td></td>
<td>CLR</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>0110</td>
<td>ACK</td>
<td>F</td>
<td>SYN</td>
<td>V</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>F</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0111</td>
<td>BEL</td>
<td>G</td>
<td>ETO</td>
<td>W</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>G</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>BS</td>
<td>CAN</td>
<td>X</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>H</td>
<td>X</td>
<td>HOME</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>HT</td>
<td>EM</td>
<td>Y</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>I</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>LF</td>
<td>SUB</td>
<td>Z</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>J</td>
<td>Z</td>
<td>EOS</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>VT</td>
<td>EBC</td>
<td>I</td>
<td>+</td>
<td>K</td>
<td>[</td>
<td>K</td>
<td></td>
<td>EL</td>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>FF</td>
<td>FS</td>
<td></td>
<td>L</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1101</td>
<td>CR</td>
<td>GS</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td>SO</td>
<td>RS</td>
<td></td>
<td>N</td>
<td>\</td>
<td>N</td>
<td>A</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>SI</td>
<td>US</td>
<td></td>
<td>?</td>
<td>O</td>
<td>-</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: Shaded blocks — functions terminal will react to. Others can be generated but are ignored upon receipt.
Appendix 7.5 Character Generator

As previously mentioned, the character generator used in this terminal is a 2716 EPROM. A 1K by 8 device would have been sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. A custom character set could have been stored in the second 1K bytes of the 2716. Any of the free I/O pins on the 8051 could have been used to switch between the character sets.

The three low-order line count outputs (LC0-LC2) from the 8276 are connected to the three low-order address lines of the character generator. The CC0-CC6 output lines are connected to the A3-A9 lines of the character generator.

The output of the character generator is loaded into the shift register. The serial output of the shift register is the video output to the CRT.

Let’s assume that the letter “E” is to be displayed. The ASCII code for “E” (45H) is presented to the address lines A2-A9 of the character generator. The scan lines (LC0-LC2) will now count from 0 to seven to form the character as shown in Figure 7.5.0. The same procedure is used to form all 128 possible characters. For reference Appendix 7.6 contains the HEX dump of the character generator used in this terminal.

45H = 01000101
Address to Prom = 01000101
= 228H – 22FH
Depending on state of Scan lines.

Character generator output

<table>
<thead>
<tr>
<th>Rom Address</th>
<th>Rom Hex Output</th>
<th>Bit Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>228H</td>
<td>3E</td>
<td></td>
</tr>
<tr>
<td>229H</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>22AH</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>22BH</td>
<td>0E</td>
<td></td>
</tr>
<tr>
<td>22CH</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>22DH</td>
<td>02</td>
<td></td>
</tr>
<tr>
<td>22EH</td>
<td>3E</td>
<td></td>
</tr>
<tr>
<td>22FH</td>
<td>00</td>
<td></td>
</tr>
</tbody>
</table>

Bits 0, 6 and 7 are not used.
*note bit output is backward from convention.

Figure 7.5.0 Character Generator
Appendix 7.7 Composite Video

In this design it was assumed that the CRT monitor required a separate horizontal drive, vertical drive, and video input. Many monitors require a composite video signal. The schematic shown in Figure 7.7.0 illustrate how to generate a composite video from the output of the 8276.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the horizontal and vertical timing is used to center the display. The 7486 is used to mix the vertical and horizontal retrace. Q1 mix the video and retrace signals along with providing the proper D.C. levels.

Figure 7.7.0 Composite Video
Appendix 7.8 Software Documentation

/***************************************************************************/
******************************************************************************/
******************************************************************************/
**** SOFTWARE DOCUMENTATION FOR THE 8051
******************************************************************************/
******************************************************************************/
******** TERMINAL CONTROLLER APPLICATION NOTE
******************************************************************************/
******************************************************************************/
******************************************************************************/
MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

8051 WR AND READ DISPLAY RAM— ADDRESS 1000H TO 17CH
8051 WR DISPLAY RAM TO THE 8276— ADDRESS 1800H TO 1FCCH
8276 COMMAND ADDRESS— ADDRESS 001H
8276 PARAMETER ADDRESS— ADDRESS 0000H
8276 STATUS REGISTER— ADDRESS 0001H

MEMORY MAP FOR READING THE KEYBOARD (USING MOVC):

KEYBOARD ADDRESS— ADDRESS 10FFH TO 17FFH

/***************************************************************************/
******************************************************************************/
******************************************************************************/
******** START MAIN PROGRAM ********
******************************************************************************/
******************************************************************************/
/***************************************************************************/
/* BEGIN BY PUTTING THE ASCII CODE FOR BLANK IN THE DISPLAY RAM*/
 INIT:
{FILL 2000 LOCATIONS IN THE DISPLAY RAM WITH SPACES (ASCII 20H)}

/* INITIALIZE POINTERS, RAM BITS, ETC. */
[INITIALIZE POINTERS AND FLAGS]
[INITIALIZE TOP OF THE CRT DISPLAY "LINE0"=1800H]
[INITIALIZE 8276 BUFFER POINTER "RASTER" =1800H]
[INITIALIZE DISPLAY RAM$POINTER=0000H]

/* INITIALIZE THE 8276 */
{RESET THE 8276}
[INITIALIZE 8276 TO 80 CHARACTER/ROW]
[INITIALIZE 8276 TO 25 ROWS PER FRAME]
[INITIALIZE 8276 TO 10 LINES PER ROW]
[INITIALIZE 8276 TO NON-BLINKING UNDERLINE CURSER]
[INITIALIZE CURSER TO HOME POSITION (00,00) (UPPER LEFT HAND CORNER)]
[START DISPLAY]
[ENABLE 8276 INTERRUPT]

/* SET UP 8051 INTERRUPTS AND PRIORITIES */
[SERIAL PORT HAS HIGHEST INTERRUPT PRIORITY]
[EXTERNAL INTERRUPTS ARE EDGE SENSITIVE]
[ENABLE EXTERNAL INTERRUPTS]
PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT WILL BE TRANSMITTED TO THE HOST COMPUTER. /*

SCANNER:

{ENABLE 8051 GLOBAL INTERRUPT BIT}

/* PROGRAMMABLE DELAY FOR THE CURSOR BLINK */

IF {30 VERTICAL RETRACE INTERRUPTS HAVE OCCURRED (CURSES$COUNT=1FH)} THEN
  {COMPLEMENT CURSES$COUNT}
  {CLEAR CURSES$COUNT}
  IF {CURSOR IS TO BE OFF (CURSES$ON=0)} THEN {MOVE CURSOR OFF THE SCREEN}
  CALL LOADCURSOR;
END;

IF {THE LOCAL$LINE SWITCH HAS CHANGED STATE} THEN
  DO;
    IF {IN LOCAL MODE} THEN {DISABLE SERIAL PORT INTERRUPT}
    ELSE CALL CHECK$BAUD$RATE;
  END;

DO WHILE {IN BETWEEN VERTICAL REFRESHES}
  IF {THE FIFO HAS A CHARACTER TO PROCESS (SERIAL$INT=1)} THEN CALL DECIPHER;
END;

CALL READER;

IF {THE PRESENT PRESSED KEY IS EQUAL TO THE LAST KEY PRESSED AND VALID=1} THEN
  CALL AUTO$REPEAT;
ELSE
  DO;
    IF {A KEY IS PRESSED BUT NOT THE SAME AS THE LAST KEYBOARD SCAN} THEN
      DO;
        IF {ONLY ONE KEY IS PRESSED} THEN
          {GET THE ASCII CODE FOR IT}
          {SET NEWS$KEY AND VALID FLAGS}
        ELSE
          {RESET VALID AND NEWS$KEY FLAGS}
        END;
      ELSE
        {THE KEYBOARD MUST NOT HAVE A KEY PressED SO RESET VALID$KEY AND NEWS$KEY FLAGS}
      END;
    GOTO SCANNER;
  END;
ENDIF;

PROCEDURE AUTO$REPEAT: THIS PROCEDURE WILL PERFORM AN AUTO REPEAT FUNCTION BY TRANSMITTING A CHARACTER EVERY OTHER TIME THIS ROUTINE IS CALLED. THE AUTO REPEAT FUNCTION IS ACTIVATED AFTER A FIXED DELAY PERIOD AFTER THE FIRST CHARACTER IS SENT. /*

AUTO$REPEAT:

IF {THE KEY PRESSED IS NEW (NEWS$KEY=1)} THEN
  DO;
    {CLEAR THE DIVIDE BY TWO COUNTER "TRANSMIT$TOGGLE"}
    {INITIALIZE THE DELAY COUNTER "TRANSMIT$COUNT" TO 0DH}
    CALL Transmit;
    {CLEAR NEWS$KEY} /* FIRST CHARACTER */
  END;
ELSE
  DO;
    {CLEAR NEWS$KEY} /* FIRST CHARACTER */
  END;
ELSE
   DO;
   IF {TRANSMIT$COUNT IS NOT EQUAL TO 0} THEN
      DO;
         [INCREMENT TRANSMIT$COUNT]
         IF TRANSMIT$COUNT=OFF THEN
            DO;
               CALL TRANSMIT;
               [CLEAR TRANSMIT$COUNT]
            END;
         END;
      END;
   ELSE
      DO;
         [TURN THE CURSOR ON DURING THE AUTO REPEAT FUNCTION]
         IF TRANSMIT$TOGGLE = 1 THEN
            CALL TRANSMIT;
            [COMPLEMENT TRANSMIT$TOGGLE]
         END;
      END;
   END AUTO$REPEAT;

PROCEDURE TRANSMIT;
PROCEDURE;
   IF [THE TERMINAL IS ON-LINE] THEN
      DO;
         [WAIT UNTIL THE CLEAR$TO$SEND LINE IS LOW AND UNTIL THE 8051 SERIAL PORT TX IS NOT BUSY (TRANSMIT$INT=1)]
         [TRANSMIT THE ASCII CODE]
         [CLEAR THE FLAG "TRANSMIT$INT". THE SERIAL PORT SERVICE ROUTINE WILL SET THE FLAG
         WHEN THE SERIAL PORT IS FINISHED TRANSMITTING]
      END;
   ELSE [THE TERMINAL IS IN THE LOCAL MODE]
      DO;
         [PUT THE ASCII CODE IN THE FIFO]
         [INCREMENT THE FIFO POINTER]
         [SET SERIAL$INT]
      END;
   END TRANSMIT;
/*    PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER'S MESSAGES AND DETERMINES    WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE    THE PROCEDURE THEN ACTS ACCORDINGLY */

DECIPHER:
START$DECIPHER:

VALID$RECEPTION=0;
DO WHILE {THE FIFO IS NOT EMPTY AND THE CHARACTER IS DISPLAYABLE}
  RECEIVE={ASCII CODE}
  CALL DISPLAY;
  {NEXT CHARACTER}
END;

IF {CHARACTERS WERE DISPLAYED} THEN
  {DISABLE SERIAL PORT INTERRUPT}
  {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
  {SET THE "VALID$RECEPTION" FLAG}

IF {THE FIFO IS EMPTY} THEN {CLEAR THE "SERIAL$INT FLAG AND RETURN"}

IF {THE NEXT CHARACTER IS AN "ESC" CODE} THEN
DO:
  {LOOK AT THE CHARACTER IN THE FIFO AFTER THE ESC CODE AND CALL THE CORRECT SUBROUTINE}
  
  CALL UP$CURSOR; /* ESC A */
  CALL DOWN$CURSOR; /* ESC B */
  CALL RIGHT$CURSOR; /* ESC C */
  CALL LEFT$CURSOR; /* ESC D */
  CALL CLEAR$SCREEN; /* ESC E */
  CALL MOVE$CURSOR; /* ESC F */
  
  CALL HOME; /* ESC H */
  CALL ERASE$FROM$CURSOR$TO$END$OF$SCREEN; /* ESC J */
  CALL BLINE; /* ESC K */

  {DISABLE THE SERIAL PORT INTERRUPT}
  {MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
  {ENABLE THE SERIAL PORT INTERRUPT}
  {SET THE "VALID$RECEPTION" FLAG}

  IF {THE FIFO IS EMPTY} THEN {CLEAR THE "SERIAL$INT FLAG AND RETURN"}
END;
IF {THE NEXT CHARACTER IS A CONTROL CODE} THEN
DO;
{CALL THE RIGHT SUBROUTINE}

CALL LEFT$CURSOR;
/* CTL H */

CALL LINESPEED;
/* CTL J */

CALL CLEAR$SCREEN;
/* CTL L */

CALL CARRIAGES$RETURN;
/* CTL M */

{DISABLE THE SERIAL PORT INTERRUPT}
{MOVE THE REMAINING CONTENTS OF THE FIFO UP TO THE BEGINNING OF THE FIFO}
{ENABLE THE SERIAL PORT INTERRUPT}
{SET THE "VALID$RECEPTION" FLAG}
END;

IF {NO VALID CODE WAS RECEIVED ("VALID$RECEPTION" IS 0)} THEN
{THROW THE CHARACTER OUT AND MOVE THE REMAINING CONTENTS OF THE FIFO}
{UP TO THE BEGINNING}

IF {THE FIFO IS EMPTY} THEN {CLEAR THE SERIAL$INT FLAG AND RETURN}

END DECIPHER;

/
*/
PROCEDURE DISPLAY: THIS PROCEDURE WILL TAKE THE BYTE IN RAM LABELED
RECEIVE AND PUT IT INTO THE DISPLAY RAM. */

DISPLAY:

{PUT INTO THE DISPLAY RAM LOCATION POINTED TO BY "DISPLAY$RAM$POINTER
THE CONTENTS OF RECEIVE]

IF {THE END OF THE DISPLAY MEMORY HAS BEEN REACHED} THEN
{RESET "DISPLAY$RAM$POINTER" TO THE BEGINNING OF THE RAM]
ELSE:
{INCREMENT "DISPLAY$RAM$POINTER"

IF {THE CURSOR IS IN THE LAST COLUMN OF THE CRT DISPLAY} THEN
DO;
MOVE THE CURSOR BACK TO THE BEGINNING OF THE LINE]
IF {THE NEW DISPLAY RAM LOCATION HAS A END-OF-LINE CHARACTER IN IT} THEN
CALL FILL;

IF {THE CURSOR IS ON THE LAST LINE OF THE CRT DISPLAY} THEN
CALL SCROLL;
ELSE:
MOVE THE CURSOR TO THE NEXT LINE]
END;
ELSE:
{INCREMENT THE CURSOR TO THE NEXT LOCATION}

{TURN THE CURSOR ON }
CALL LOADCURSOR;
END DISPLAY;

10-96
/* PROCEDURE LINESFEED */

LINESFEED:
IF (THE CURSOR IS IN THE LAST LINE OF THE CRT DISPLAY) THEN
    CALL SCROLL;
ELSE
    DO;
        MOVE THE CURSOR TO THE NEXT LINE
        (TURN THE CURSOR ON)
        CALL LOAD$CURSER;
    END;
IF (THE DISPLAY$RAM$POINTER IS ON THE LAST LINE IN THE DISPLAY RAM) THEN
    MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY RAM
ELSE
    MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY RAM
IF (THE FIRST CHARACTER IN THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER) THEN
    CALL FILL;
END LINESFEED;

/* PROCEDURE SCROLL */

SCROLL:
CALL BLANK;
(DISABLE VERTICAL RETRACE INTERRUPT)
IF (THE FIRST LINE OF THE CRT CONTAINS THE LAST LINE OF THE DISPLAY MEMORY) THEN
    MOVE THE POINTER "LINE0" TO THE BEGINNING OF THE DISPLAY MEMORY
ELSE
    MOVE "LINE0" TO THE NEXT LINE IN THE DISPLAY MEMORY
{ENABLE VERTICAL RETRACE INTERRUPT}
END SCROLL;

/* PROCEDURE CLEAR SCREEN */

CLEAR$SCREEN:
CALL HOME;
CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
END CLEAR$SCREEN;
/* PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSOR TO THE 0,0 POSITION */

HOME:

- MOVE THE CURSOR POSITION TO THE UPPER LEFT HAND CORNER OF THE CRT
- TURN THE CURSOR ON
- CALL LOADCURSER;
- MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION IN THE DISPLAY RAM

END HOME;

/* PROCEDURE ERASE FROM CURSOR TO END OF SCREEN: */

ERASEFROMCURSERTOSENDOFSCREEN:

CALL BLINE;  /* ERASE CURRENT LINE */

IF (THE CURSOR IS NOT ON THE LAST LINE OF THE CRT DISPLAY) THEN
  STARTING WITH THE NEXT LINE, PUT AN END-OF-LINE CHARACTER (UFH) IN THE DISPLAY RAM LOCATIONS THAT CORRESPOND TO THE BEGINNING OF THE CRT DISPLAY LINES UNTIL THE BOTTOM OF THE CRT SCREEN HAS BEEN REACHED
END;

END ERASEFROMCURSERTOSENDOFSCREEN;

/* PROCEDURE MOV$CURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR */

IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL READ THE NEXT TWO BYTES TO DETERMINE WHERE TO MOVE THE CURSOR. THE FIRST BYTE IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */

MOV$CURSER:

- WAIT UNTIL THE FIFO HAS RECEIVED THE NEXT TWO CHARACTERS
- MOVE THE CURSOR TO THE LOCATION SPECIFIED IN THE ESCAPE SEQUENCE
- MOVE THE DISPLAY$RAM$POINTER TO THE CORRECT LOCATION

IF THE FIRST CHARACTER IN THE NEW LINE HAS AN END-OF-LINE CHARACTER THEN
  CALL FILL;
END;

- DISABLE THE SERIAL PORT INTERRUPT
- MOVE THE REMAIN CONTENTS OF THE FIFO UP TWO LOCATIONS IN MEMORY
- DECREMENT THE FIFO BY TWO
- ENABLE THE SERIAL PORT INTERRUPT

END MOV$CURSER;

/* PROCEDURE LEFT$CURSER: THIS PROCEDURE MOVES THE CURSOR LEFT ONE COLUMN BY SUBTRACTING 1 OF THE CURSOR COLUMN RAM LOCATION THEN CALL LOAD CURSER */

LEFT$CURSER:

IF (THE CURSOR IS NOT IN THE FIRST LOCATION OF A LINE) THEN
  DO;
    MOVE THE CURSOR LEFT BY ONE LOCATION
    TURN THE CURSOR ON
    CALL LOADCURSER;
    DECREMENT THE DISPLAY$RAM$POINTER BY ONE
  END;
END LEFT$CURSER;
/* PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
   BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */
RIGHT$CURSER:
IF {THE CURSER IS NOT IN THE LAST POSITION OF THE CRT LINE} THEN
  DO:
  {MOVE THE CURSER RIGHT BY ONE LOCATION}
  {TURN THE CURSER ON}
  CALL LOAD$CURSER;
  {INCREMENT THE DISPLAY$RAM$POINTER BY ONE}
  END;
END RIGHT$CURSER;

/* PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
   BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */
UP$CURSER:
IF {THE CURSER IS NOT ON THE FIRST LINE OF THE CRT DISPLAY} THEN
  DO:
  {MOVE THE CURSER UP ONE LINE}
  {TURN ON THE CURSER}
  CALL LOAD$CURSER;
  {INCREASE THE DISPLAY$RAM$POINTER TO THE LAST LINE OF DISPLAY MEMORY}
  IF {THE DISPLAY$RAM$POINTER IS IN THE FIRST LINE OF DISPLAY MEMORY} THEN
    MOVE THE DISPLAY$RAM$POINTER UP ONE LINE IN DISPLAY MEMORY
  ELSE
    MOV THE DISPLAY$RAM$POINTER UP ONE LINE IN DISPLAY MEMORY
  END;
  IF {THE FIRST LOCATION OF THE NEW LINE CONTAINS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;
  END;
END UP$CURSER;

/* PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
   BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */
DOWN$CURSER:
IF {THE CURSER IS NOT ON THE LAST LINE OF THE CRT DISPLAY} THEN
  DO:
  {TURN THE CURSOR ON}
  {MOVE THE CURSOR TO THE NEXT LINE}
  CALL LOAD$CURSER;
  IF {THE DISPLAY$RAM$POINTER IS NOT ON THE LAST LINE OF THE DISPLAY MEMORY} THEN
    MOVE THE DISPLAY$RAM$POINTER TO THE NEXT LINE IN THE DISPLAY MEMORY
  ELSE
    MOVE THE DISPLAY$RAM$POINTER TO THE FIRST LINE IN THE DISPLAY MEMORY
  END;
  IF {THE FIRST CHARACTER IN THE NEW LINE IS AN END-OF-LINE CHARACTER} THEN
    CALL FILL;
  END;
END DOWN$CURSER;
/* PROCEDURE CARRIAGE$RETURN */

CARRIAGE$RETURN:

{MOVE THE DISPLAY$RAM$POINTER TO THE BEGINNING OF THE CURRENT LINE IN THE DISPLAY MEMORY}
{MOVE THE CURSOR TO THE BEGINNING OF THE CURRENT LINE OF THE CRT DISPLAY}
{TURN THE CURSOR ON}
CALL LOAD$CURSER;
END CARRIAGE$RETURN;

/* PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND LOADS IT INTO THE 8276 CURSER REGISTER. */

LOAD$CURSER:

PROCEDURE;

IF [THE CURSER IS ON] THEN
{MOVE THE CURSER BACK ONTO THE CRT DISPLAY}
{DISABLE BUFFER INTERRUPT}
{WRITE TO THE 8276 CURSER REGISTERS THE X,Y LOCATIONS}
{ENABLE BUFFER INTERRUPT}

END LOAD$CURSER;

/* PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

CHECK$BAUD$RATE:

{SET TIMER 1 TO MODE 1 AND AUTO RELOAD}
{TURN TIMER ON}
{ENABLE SERIAL PORT INTERRUPT}
{READ BAUD RATE SWITCHES AND SET UP RELOAD VALUE} ;

; /* 00 IS NOT ALLOWED */
TH1=040H; /* 150 BAUD */
TH1=0A0H; /* 300 BAUD */
TH1=0D0H; /* 600 BAUD */
TH1=0E8H; /* 1200 BAUD */
TH1=0F4H; /* 2400 BAUD */
TH1=0F8H; /* 4800 BAUD */
TH1=0FDH; /* 9600 BAUD */

END CHECK$BAUD$RATE;
/* PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL MEMORY ARRAY CALLED CURRENT$KEY */

READER:

[INITIALIZE FLAGS "KEY0"=0, "SAME"=1, 0 COUNTER=0]

DO UNTIL [ALL 8 KEYBOARD SCAN LINES ARE READ]

[READ KEYBOARD SCAN]

IF [NO KEY WAS Pressed] THEN

[INCREMENT 0 COUNTER]

ELSE

IF [THE KEY PRESSED WAS NOT THE SAME KEY THAT WAS PRESSED THE LAST TIME THE KEYBOARD WAS READ] THEN

[CLEAR "SAME" AND WRITE NEW SCAN RESULT TO CURRENT$KEY RAM ARRAY]

END;

IF [ALL 8 SCANS DIDN'T HAVE A KEY PRESSED (0 COUNTER=8)] THEN

[SET KEY0, AND CLEAR SAME]

END READER;

;/* PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 WITH SPACES (20H ASCII) DURING THE SCROLL Routines.*/

BLANK:

DO I= [BEGINNING OF THE CRT DISPLAY (LINE0)] TO [LINE0 + 50H]

[DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)]

NEXT I

END;

END BLANK;

;/* PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSOR TO THE END OF THE DISPLAY LINE */

BLINE:

DO I= [CURRENT CURSOR POSITION ON CRT DISPLAY] TO [END OF ROW]

[DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)]

NEXT I

END;

END BLINE;

;/* PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS A DISPLAY LINE WITH SPACES*/

FILL:

DO I= [BEGINNING OF THE LINE THAT THE CURSOR IS ON] TO [END OF THE ROW]

[DISPLAY RAM POINTED TO BY "I" = SPACE (ASCII 20H)]

NEXT I

END;

END FILL;
Appendix 7.9 Software Listings

PL/M-51 COMPILER

ISIS-II PL/M-51 V1.1
COMPILER INVOKED BY: PL/M51:PL:CRTPLM.SRC

$OPTIMIZE(1)
$NOINVECTOR
$RM(LARGE)

******************************************************************************
******
****** PLM51 SOFTWARE FOR THE 8051 TERMINAL
****** CONTROLLER APPLICATION NOTE
******
******************************************************************************

MEMORY MAP ASSOCIATED WITH PERIPHERAL DEVICES (USING MOVX):

8051 WR AND READ DISPLAY RAM—ADDRESS 1000H TO 17CH
8051 WR DISPLAY RAM TO THE 8276—ADDRESS 1800H TO 1FCCH
8276 COMMAND ADDRESS—ADDRESS 0001H
8276 PARAMETER ADDRESS—ADDRESS 000OH
8276 STATUS REGISTER—ADDRESS 0001H

MEMORY MAP FOR READING THE KEYBOARD (USING MOVY):

KEYBOARD ADDRESS—ADDRESS 10FPH TO 17FFH

THE FOLLOWING SOFTWARE SWITCHES MUST BE SET ACCORDING TO THE TYPE OF
KEYBOARD THAT IS GOING TO BE USED.

SW1—SET WHEN USING AN UNDECODED KEYBOARD IS TO BE USED
SW2—SET WHEN USING A DECODED OR A SERIAL TYPE OF KEYBOARD

PROGRAMS TO LINK TOGETHER FOR WORKING SYSTEMS:

UNDECODED KEYBOARD—CRTPLM.OBJ,CRTPLM.OBJ,KEYBD.OBJ,PLM51.LIB
DECODED KEYBOARD—CRTPLM.OBJ,CRTPLM.OBJ,DECODE.OBJ,PLM51.LIB
DETACHED KEYBOARD—CRTPLM.OBJ,CRTPLM.OBJ,DETACH.OBJ,PLM51.LIB

*/
$SET (SW1)
$RESET (SW2)
DECLARE LITERALS

DECLARE LLC LITERALLY 'LOCAL:\LINE\ CHANGE';
DECLARE REG LITERALLY 'REGISTER';
DECLARE CURRENTSKY LITERALLY 'CURSKY';
DECLARE SERIAL\$SERVICE LITERALLY 'SERBUF';
DECLARE DISPLAY\$POINTER LITERALLY 'POINT';
DECLARE SERIAL\$INT LITERALLY 'SERINT';
DECLARE TRANSMIT\$INT LITERALLY 'TRANINT';
DECLARE CURSOR\$COLUMN LITERALLY 'CURSER';
DECLARE SERIAL\$INT LITERALLY 'SCANT';
DECLARE SCAN\$INT LITERALLY 'SCAN';

REGISTER DECLARATIONS FOR THE 8051

BYTE REGISTERS

DECLARE
P0 BYTE AT (80H) REG,
P1 BYTE AT (90H) REG,
P2 BYTE AT (A0H) REG,
P3 BYTE AT (B0H) REG,
PSW BYTE AT (OD0H) REG,
ACC BYTE AT (0E0H) REG,
B BYTE AT (0F0H) REG,
SP BYTE AT (81H) REG,
DPL BYTE AT (82H) REG,
DPH BYTE AT (83H) REG,
PCON BYTE AT (87H) REG,
TCON BYTE AT (88H) REG,
TMOD BYTE AT (89H) REG,
TL0 BYTE AT (8AH) REG,
TL1 BYTE AT (8BH) REG,
TH0 BYTE AT (8CH) REG,
TH1 BYTE AT (8DH) REG,
IE BYTE AT (90H) REG,
IP BYTE AT (91H) REG,
SCON BYTE AT (98H) REG,
SEBUF BYTE AT (99H) REG;
SELECT

********** BIT REGISTERS **********

DECLARE

PSW BITS

14 1

CY BIT AT (007H) REG,
AC BIT AT (006H) REG,
P0 BIT AT (005H) REG,
N1 BIT AT (004H) REG,
N0 BIT AT (003H) REG,
OV BIT AT (002H) REG,
P BIT AT (001H) REG,

TCON BITS

TF1 BIT AT (8FH) REG,
TK1 BIT AT (8EH) REG,
TF0 BIT AT (8DH) REG,
TR0 BIT AT (8CH) REG,
IE1 BIT AT (8BH) REG,
IT1 BIT AT (8AH) REG,
IE0 BIT AT (89H) REG,
IT0 BIT AT (88H) REG,

IE BITS

EA BIT AT (0AFH) REG,
ES BIT AT (0AH) REG,
ET1 BIT AT (09H) REG,
EK1 BIT AT (08H) REG,
ETO BIT AT (07H) REG,
EEK BIT AT (06H) REG,

IP BITS

PS BIT AT (0BCH) REG,
PT1 BIT AT (0BBH) REG,
PX1 BIT AT (0B9H) REG,
PTO BIT AT (0B8H) REG,

P3 BITS

RD BIT AT (0B7H) REG,
WR BIT AT (0B6H) REG,
T1 BIT AT (0B5H) REG,
T0 BIT AT (0B4H) REG,
INT1 BIT AT (0B3H) REG,
INT0 BIT AT (0B2H) REG,
TXD BIT AT (0B1H) REG,
RD0 BIT AT (0B0H) REG,

SCon BITS

SM0 BIT AT (9FH) REG,
SM1 BIT AT (9EH) REG,
SM2 BIT AT (9DH) REG,
REN BIT AT (9CH) REG,
TB8 BIT AT (9BH) REG,
RB8 BIT AT (9AH) REG,
TI BIT AT (99H) REG,
RI BIT AT (98H) REG,
AP-223

PL/M-51 COMPILER  CRYCONTROLLER

SELECT
$IF SWL
/***************** DECLARE CONSTANTS****************/

15 1 DECLARE LOWSCAN(16) STRUCTURE
(KEY (8) BYTE) CONSTANT

('890-'5CH,5EH,08H,00H,
/* SCAN 0, SHIFT KEY =0; 8,9,0,-,", BACK SPACE */
'uiop',5BH,'@',0AH,7FH,
/* SCAN 1, SHIFT =0; u,i,o,p,[,], LINE FEED, DELETE */
'jkl;',00H,00H,','
/* SCAN 2, SHIFT =0; j,k,l,:, RETURN, 7 */
'm',2CH,'.',00H,'/',00H,00H,00H,
/* SCAN 3, SHIFT =0; m,COMMA,:, */
00H,'azxcvbn',
/* SCAN 4, SHIFT =0; a,z,x,c,v,b,n */
'y',00H,00H,' dfgh',
/* SCAN 5, SHIFT =0; y, SPACE, d,f,g,h */
09H,'qwert',00H,
/* SCAN 6, SHIFT =0; t,a,b,q,w,e,r,t */
1BH,'123456',00H,
/* SCAN 7, SHIFT =0; ESC,1,2,3,4,5,6 */
2BH,28H,00H,=)',7CH,7EH,08H,00H,
/* SCAN 0, SHIFT =1; (,),=,", BACK SPACE */
'UIOP',00H,00H,0AH,7FH,
/* SCAN 1, SHIFT =1; U,I,O,P, LINE FEED, DELETE */
'JLKL*',00H,00H,27H,
/* SCAN 2, SHIFT =1; J,K,L,*, RETURN, 1 */
'M<>',00H,3FH,00H,00H,00H,00H,
/* SCAN 3, SHIFT =1; M,<>,? */
00H,'AZXCVBN',
/* SCAN 4, SHIFT =1; A,Z,X,C,V,B,N */
'y',00H,00H,' dfgh',
/* SCAN 5, SHIFT =1; y, SPACE, D,F,G,H */
09H,'QWERT',00H,
/* SCAN 6, SHIFT =1; TAB, Q,W,E,R,T */
1BH,'!*%&',00H);
/* SCAN 7, SHIFT =1; ESC,!,",%,& */
$ENDIF

10-105
DECLARE

$IF SW2

INPUT

$IFDEF SW2

\[\text{BIT AT (064H)} \quad \text{REG,}\]

$ENDIF

\[\text{BIT AT (095H)} \quad \text{REG,}\]

SHIFT$KEY

\[\text{BIT AT (096H)} \quad \text{REG,}\]

CONTROL$KEY

\[\text{BIT AT (097H)} \quad \text{REG,}\]

$ENDIF

LOCAL$LINE

\[\text{BIT AT (089H)} \quad \text{REG,}\]

CLEAR$TO$SEND

\[\text{BIT AT (093H)} \quad \text{REG,}\]

DATA$TERMINAL$READY

\[\text{BIT AT (094H)} \quad \text{REG,}\]

$IFDEF SW1

SAME,

\[\text{VALID$KEY,}\]

KEY0,

\[\text{LAST$SHIFT$KEY,}\]

\[\text{LAST$CONTROL$KEY,}\]

\[\text{LAST$CAP$LOCK,}\]

$ENDIF

$IFDEF SW2

KC$VLG,

\[\text{SYNC,}\]

\[\text{BYFIN,}\]

\[\text{KB$INT,}\]

\[\text{ERROR,}\]

$ENDIF

\[\text{NEW$KEY,}\]

\[\text{TRANSMIT$TOGGLE,}\]

\[\text{CURSER$SON,}\]

\[\text{SERIAL$INT,}\]

\[\text{SCAN$INT,}\]

\[\text{TRANSMIT$INT,}\]

\[\text{ESCSEQ,}\]

\[\text{VALID$RECEPTION,}\]

\[\text{LLC,}\]

\[\text{ENSP)} \quad \text{BIT PUBLIC;}\]
DECLARE (I, J, K, ASCII$KEY, TRANSMIT$COUNT, TEMP, SHIFT, CURSOR$COL, CURSOR$COLUMN, CURSOR$ROW, CURSOR$COUNT, FIFO, RECEIVE) BYTE PUBLIC;

$IF SW1
DECLARE LAST$KEY(8) BYTE PUBLIC;
$ENDIF

$IFDEF SW2
DECLARE LAST$KEY(2) BYTE PUBLIC;
$ENDIF

DECLARE SERIAL(16) BYTE PUBLIC;

DECLARE DISPLAY$RAM(7CFH) BYTE AT(1000H) AUXILIARY;

DECLARE
PAR$RAM$ADDRESS BYTE AT(0000H) AUXILIARY,
COMMAND$ADDRESS BYTE AT(0001H) AUXILIARY;

DECLARE (DISPLAY$RAM$POINTER, RASTER, LINES, L) WORD PUBLIC;
PROCEDURE READER: THIS PROCEDURE IS WRITTEN IN ASSEMBLY LANGUAGE. THE EXTERNAL PROCEDURE SCANS THE 8 LINES OF THE KEYBOARD AND READS THE RETURN LINES. THE STATUS OF THE 8 RETURN LINES ARE THEN STORED IN INTERNAL MEMORY ARRAY CALLED CURRENTKEY. THE PROCEDURE CONTROLS 2 STATUS FLAGS; KEY0 AND SAME. KEY0 IS SET IF ALL 8 SCANS READ NO KEY WAS PRESSED. IF ALL 8 SCANS ARE THE SAME AS THE LAST READING OF THE KEYBOARD, THEN SAME IS SET. */

24 2 READER: PROCEDURE EXTERNAL;
25 1 END READER;

PROCEDURE BLANK: THIS EXTERNAL PROCEDURE FILLS LINE0 SCAN WITH SPACES (20H ASCII) DURING THE SCROLL ROUTINES.*/

26 2 BLANK: PROCEDURE EXTERNAL;
27 1 END BLANK;

PROCEDURE BLINE: THIS EXTERNAL PROCEDURE BLANKS FROM THE CURSOR TO THE END OF THE DISPLAY LINE */

28 2 BLINE: PROCEDURE EXTERNAL;
29 1 END BLINE;

PROCEDURE FILL: THIS EXTERNAL PROCEDURE FILLS THE CURSOR LINE WITH SPACES*/

30 1 FILL:
PROCEDURE EXTERNAL;
31 1 END FILL;
PL/M-51 COMPIILER  CRTCNTROLLER

$EXECT

/* PROCEDURE CHECK BAUD RATE: THIS PROCEDURE READS THE THREE PORT PINS ON P1 AND SETS UP THE SERIAL PORT FOR THE SPECIFIED BAUD RATE */

32 1 CHECK$BAUD$RATE:
PROCEDURE;
33 2 SCON=70H; /* MODE 1 ENABLE RECEPTION*/
34 2 TMOD=TMOD OR 20H; /* TIMER 1 AUTO RELOAD */
35 2 TR1=1; /* TIMER 1 ON */
36 2 ES=1; /* ENABLE SERIAL INTERRUPT*/
37 2 ENSP=1; /* SERIAL INTERRUPT MASK FLAG */
38 3 DO CASE (P1 AND 07H);
39 3 1; /* 00 IS NOT ALLOWED */
40 3 TH1=040H; /* 150 BAUD */
41 3 TH1=0A0H; /* 300 BAUD */
42 3 TH1=0D0H; /* 600 BAUD */
43 3 TH1=0E8H; /* 1200 BAUD */
44 3 TH1=0F4H; /* 2400 BAUD */
45 3 TH1=0F8H; /* 4800 BAUD */
46 3 TH1=0FH; /* 9600 BAUD */
47 3 END;
48 1 END CHECK$BAUD$RATE;

/* PROCEDURE LOAD CURSER: LOAD CURSER TAKES THE VALUE HELD IN RAM AND LOADS IT INTO THE 8276 CURSER REGISTERS. */

49 1 LOAD$CURSER:
PROCEDURE;
50 2 IF CURSER$CON=1 THEN
51 2 CURSER$COL=CURSER$COLUMN;
52 2 EX1=0; /* DISABLE BUFFER INTERRUPT */
53 2 COMMAND$ADDRESS=80H; /* INITIALIZE CURSER COMMAND */
54 2 PARAMETER$ADDRESS=CURSER$COL;
55 2 PARAMETER$ADDRESS=CURSER$ROW;
56 2 EX1=1; /* ENABLE BUFFER INTERRUPT */
57 1 END LOAD$CURSER;

/* PROCEDURE CARRIAGE$RETURN */

58 1 CARRIAGE$RETURN:
PROCEDURE;
59 2 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-CURSER$COLUMN;
60 2 CURSER$COLUMN=0;
61 2 CURSER$CON=1;
62 2 CALL LOAD$CURSER;
63 1 END CARRIAGE$RETURN;

10-109
SUBJECT

/* PROCEDURE DOWN CURSER: THIS PROCEDURE MOVES THE CURSER DOWN ONE ROW
   BY ADDING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

64   DOWNCURSER:
       PROCEDURE;
65   IF CURSER$ROW < 18H THEN
66      DO;
67      CURSER$ROW=CURSER$ROW + 1;
68      CALL LOADCURSER;
69      IF DISPLAY$RAM$POINTER < 780H THEN
70      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER + 50H;
71      ELSE
72      DISPLAY$RAM$POINTER=(DISPLAY$RAM$POINTER-780H);
73      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
74      IF DISPLAY$RAM(L)=OF1H THEN /* LOOK FOR END OF LINE*/
75      DO; /* LINE CHARACTER */
76      CALL FILL; /* IF TRUE FILL LINE*/
77      DISPLAY$RAM(L)=20H; /* WITH SPACES */
78      END;
79      END;
80   END DOWNCURSER;

/* PROCEDURE UP CURSER: THIS PROCEDURE MOVES THE CURSER UP ONE ROW
   BY SUBTRACTING 1 TO THE CURSER ROW RAM LOCATION THEN CALL LOAD CURSER */

81   UPSCURSER:
       PROCEDURE;
82   IF CURSER$ROW > 0 THEN
83      DO;
84      CURSER$ROW=CURSER$ROW - 1;
85      CALL LOADCURSER;
86      IF DISPLAY$RAM$POINTER<50H THEN
87      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+780H;
88      ELSE
89      DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-50H;
90      L=DISPLAY$RAM$POINTER-CURSER$COLUMN;
91      IF DISPLAY$RAM(L)=OF1H THEN /* LOOK FOR END OF LINE*/
92      DO; /* CHARACTER */
93      CALL FILL; /* IF TRUE FILL WITH */
94      DISPLAY$RAM(L)=20H; /* SPACES */
95      END;
96      END;
97   END UPSCURSER;

10-110
PL/M-51 COMPILER  CRTCONTROLLER

$EXECT

/* PROCEDURE RIGHT CURSER: THIS PROCEDURE MOVES THE CURSER RIGHT ONE COLUMN
   BY ADDING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

98  1   RIGHT$CURSER:
   PROCEDURE;
99  2   IF CURSER$COLUMN < 4FH THEN
100  3       DO;
101  3       CURSER$COLUMN=CURSER$COLUMN + 1;
102  3       CURSOR$ON=1;
103  3       CALL LOAD$CURSER;
104  3       DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER +1;
105  3       END;
106  1   END RIGHT$CURSER;

/* PROCEDURE LEFT CURSER: THIS PROCEDURE MOVES THE CURSER LEFT ONE COLUMN
   BY SUBTRACTING 1 TO THE CURSER COLUMN RAM LOCATION THEN CALL LOAD CURSER */

107  1   LEFT$CURSER:
   PROCEDURE;
108  2   IF CURSER$COLUMN >0 THEN
109  3       DO;
110  3       CURSER$COLUMN=CURSER$COLUMN - 1;
111  3       CURSOR$ON=1;
112  3       CALL LOAD$CURSER;
113  3       DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER -1;
114  3       END;
115  1   END LEFT$CURSER;
PL/M-51 COMPI L ER C R I CONTROL L ER

SELECT

/* PROCEDURE MOVSCURSER: THIS PROCEDURE IS USED IN CONJUNCTION WITH WORDSTAR
   IF A ESC F IS RECEIVED FROM THE HOST COMPUTER, THE TERMINAL CONTROLLER WILL
   READ THE NEXT TWO Bytes TO DETERMINE WHERE TO MOVE THE CURSER. THE FIRST BYTE
   IS THE ROW INFORMATION FOLLOWED BY THE COLUMN INFORMATION */

116 1 MOVSCURSER:
   PROCEDURE;
117 3 DO WHILE FIFO<4; /* WAIT UNTIL THE MOVSCURSER PARAMETERS*/
118 3 END; /* ARE RECEIVED INTO THE FIFO */
119 2 TEMP=CURRSE R$COLUMN;
120 2 CURSER$COLUMN=SERIAL(2);
121 2 IF CURRSE R$COLUMN=TEMP THEN
122 3 DO;
123 3 L=DISPLAY$RAM$POINTER$+ (CURRSE R$COLUMN-TEMP) *50H);
124 3 IF L>7FH THEN
125 3 DISPLAY$RAM$POINTER=L-7D0H; /* RAP AROUND TO BEGINNING */
126 3 ELSE
127 3 DISPLAY$RAM$POINTER=L;
128 2 ELSE
129 3 IF CURRSE R$COLUMN=TEMP THEN
130 4 DO;
131 4 L= (TEMP-CURRSE R$COLUMN) *50H;
132 4 IF DISPLAY$RAM$POINTER=L THEN /* IF OUT OF RAM RANGE */
133 4 DISPLAY$RAM$POINTER=(7D0H-L-DISPLAY$RAM$POINTER)) / * RAP AROUND TO END OF RAM */
134 4 ELSE
135 4 END;
136 3 END;
137 2 TEMP=CURRSE R$COLUMN;
138 2 CURRSE R$COLUMN=SERIAL(3);
139 2 IF CURRSE R$COLUMN=TEMP THEN
140 2 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER$+ (CURRSE R$COLUMN-TEMP);
141 2 ELSE
142 2 DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER$- (TEMP-CURRSE R$COLUMN);
143 2 CURRSE R$=1;
144 2 CALL LOADCURS E R;
145 2 L=DISPLAY$RAM$POINTER-CURRSE R$COLUMN;
146 2 IF DISPLAY$RAM$(L)=0FH THEN /* LOOK FOR END PO LINE CHARACTER*/
147 3 CALL FILL; /* IF TRUE FILL WITH SPACES */
148 3 IF DISPLAY$RAM$(L)=0FH THEN
149 2 END;
150 2 E$=0;
151 3 DO I=2 TO FIFO-2;
152 3 SERIAL(I)+=SERIAL(I+2);
153 3 END;
154 2 FIFO=FIFO-2;
155 2 ES=E$;
156 1 END MOVSCURSER;
PL/M-51 COMPILER CRT CONTROLLER

SUBROUTINE

/* PROCEDURE ERASE FROM CURSOR TO END OF SCREEN: */

157          ERASE$FROM$CURSOR$TO$END$OF$SCREEN:
158          PROCEDURE;
159          CALL BLINE;            /* ERASE CURRENT LINE */
160          IF CURSER$NOW < 18H THEN
161          DO:
162          L=DISPLAY$RAM$POINTER$CURSER$COLUMN+50H; /* GET NEXT LINE */
163          DO WHILE (L < 700H) AND (L <> (LINE0 AND 7FFH));
164          DISPLAY$RAM(L)=0FH;       /* ERASE UNTIL LINE0 OR */
165          L=L+50H;                    /* END OF DISPLAY RAM*/
166          END;
167          DO WHILE L <> (LINE0 AND 7FFH); /* ERASE UNTIL LINE0 */
168          DISPLAY$RAM(L)=0FH;
169          L=L+50H;
170          END;
171          END;
172          END ERASE$FROM$CURSOR$TO$END$OF$SCREEN;

/* PROCEDURE HOME: THIS PROCEDURE MOVES THE CURSOR TO THE 0,0 POSITION */

173          HOME:
174          PROCEDURE;
175          CURSER$ROW=00;
176          CURSER$COLUMN=00;
177          CALL LOAD$CURSER;
178          DISPLAY$RAM$POINTER=(LINE0 AND 7FFH);
179          END HOME;
AP-223

PL/M-51 COMPILER CRT CONTROLLER

$DEFINE

/* PROCEDE CLEAR SCREEN */

180 1 CLEARSCREEN:
   PROCEDURE;
181 2 CALL HOME;
182 2 CALL ERASE$FROM$CURSER$TO$END$OF$SCREEN;
183 1 END CLEARSCREEN;

/* PROCEDURE SCROLL */

184 1 SCROLL:
   PROCEDURE;
185 2 CALL BLANK;
186 2 EX0=0;    /* DISABLE VERTICAL REFRESH INTERRUPT */
187 2 IF LINE0=1F80H THEN
188 2   LINE0= 1800H;
189 2 ELSE
190 2   LINE0= LINE0+50H;
191 1 END SCROLL;

/* PROCEDE LINE$FEED */

192 1 LINE$FEED:
   PROCEDURE;
193 2 IF CURSER$SHOW=18H THEN
194 2   CALL SCROLL;
195 2 ELSE
196 3   CURSER$SHOW= CURSER$SHOW+1;
197 3   CURSER$CON=1;
198 3   CALL LOAD$CURSER;
199 3 END;
200 2 IF DISPLAY$RAM$POINTER >7FH THEN
201 2   DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER-780H;
202 2 ELSE
203 2   DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+50H;
204 2 IF DISPLAY$RAM$POINTER=CURSER$COLUMN THEN
205 3   DL;
206 3   CALL FILL;    /* IF TRUE FILL WITH SPACES */
207 3   DISPLAY$RAM$POINTER=20H;
208 3 END;
209 1 END LINE$FEED;

10-114
DISPLAY:
PROCEDURE;
DISPLAY$RAM(DISPLAY$RAM$POINTER)=RECEIVE;
IF DISPLAY$RAM$POINTER=7FH THEN
/* IF END OF RAM */
DISPLAY$RAM$POINTER=00H;
/* RAP AROUND TO BEGINNING */
ELSE
DISPLAY$RAM$POINTER=DISPLAY$RAM$POINTER+1;
IF CURSER$COLUMN=4FH THEN
DO;
CURSER$COLUMN=00H;
L=DISPLAY$RAM$POINTER;
IF DISPLAY$RAM(L)=0FH THEN
DO;
CALL FILL;
DISPLAY$RAM(L)=20H;
END;
IF CURSER$ROW=18H THEN
CALL SCROLL;
ELSE
CURSER$ROW=CURSER$ROW+1;
END;
ELSE
CURSER$COLUMN=CURSER$COLUMN+1;
CURSER$ROW=1;
CALL LOADCURSER;
END DISPLAY;

PL/M-51 COMPILER CRT CONTROLLER

SECT
/
* PROCEDURE DECIPHER: THIS PROCEDURE DECODES THE HOST COMPUTER’S MESSAGES AND DETERMINES
WHETHER IT IS A DISPLAYABLE CHARACTER, CONTROL SEQUENCE, OR AN ESCAPE SEQUENCE
THE PROCEDURE THEN ACTS ACCORDINGLY */

232 1 DECIPHER:
PROCEDURE;
233 2 STARTDECIPHER:
VALID$RECEPTION=0;
234 2 I=0;
235 3 DO WHILE (I<FIPO) AND (SERIAL(I)>1FH) AND (SERIAL(I)<7FH);
236 3 RECEIVE=SERIAL(I);
237 3 CALL DISPLAY;
238 3 I=I+1;
239 3 END;
240 2 IF I>0 THEN
241 3 DO;
242 3 ES=0;
243 3 K=FIPO-I;
244 4 DO J=0 TO K;
245 4 SERIAL(J)=SERIAL(I);
246 4 I=I+1;
247 4 END;
248 3 IFPO=K;
249 3 ES=ENSP;
250 3 VALID$RECEPTION=1;
251 3 END;
252 2 IF FIFO=0 THEN
253 3 DO;
254 3 SERIAL$INT=0;
255 3 GOTO END$DECIPHER;
256 3 END;
257 2 IF (SERIAL(0)=1BH) THEN
258 3 DO;
259 3 IF (ESCSSEQ=1) AND (FIPO<2) THEN
260 3 GOTO END$DECIPHER;
261 3 K=(SERIAL(1) AND 5FH)-40H;
262 3 IF (K >0FH) AND (K<0CH) THEN
263 4 DO;
264 5 DO CASE K;
265 5 ;
266 5 CALL UP$CURSOR;
267 5 CALL DOWN$CURSOR;
268 5 CALL RIGHT$CURSOR;
269 5 CALL LEFT$CURSOR;
270 5 CALL CLEAR$SCREEN;
271 5 CALL MOVE$CURSOR;
272 5 ;
273 5 CALL HOME;
274 5 ;
275 5 CALL ERASE$FROM$CURSOR$TO$END$OF$SCREEN;
276 5 CALL BLINE;
277 5 END;
278 4 END;
279 3 ES=0;
280 3 /* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
PL/M-51 COMPILER

CRTOCONTROLLER

DO I=0 TO (FIFO-2);  
  SERIAL(I)=SERIAL(I+2); /* MOVE FIFO */
END;

FIFO=FIFO-2;
ES=ENSP;
VALID$RECEPTION=1;
IF FIFO=0 THEN
DO;
  SERIAL$INT=0;
  GOTO ENDCIPHER;
END;

IF (SERIAL(0)>07H) AND (SERIAL(0)<0FH) THEN
DO;
  DO CASE (SERIAL(0) -08H);
  CALL LEFT$CURSOR; /* CTL H */
  CALL LINES$FEED; /* CTL J */
  CALL CLEAR$SCREEN; /* CTL L */
  CALL CARRIAGE$RETURN; /* CTL M */
END;

ES=0;
/* DISABLE SERIAL INTERRUPTS WHILE MOVING FIFO */
DO I=0 TO (FIFO-1);
  SERIAL(I)=SERIAL(I+1); /* MOVE FIFO */
END;

FIFO=FIFO-1;
ES=ENSP;
VALID$RECEPTION=1;
END;

IF VALID$RECEPTION=0 THEN
DO;
ES=0;
DO I=0 TO (FIFO-1); /* IF CHARACTER IS UNRECOGNIZED THEN */
  SERIAL(I)=SERIAL(I+1); /* TRASH IT */
END;
FIFO=FIFO-1;
ES=ENSP;
END;

IF FIFO=0 THEN
SERIAL$INT=0;
ENDCIPHER;
END DECIPHER:
PL/M-51 Compiler  CRController

/* Select */
#include "DDR";  

BEGIN

/* Procedure Transmit - This procedure looks at the clear to send pin for an active
low signal. Once the main computer signals the 8051 the ASCII character is put
into the serial port. */

TRANSMIT:  
PROCEDURE;

IF LOCAL$LINE =1 THEN

DO WHILE (CLEAR$TO$SEND=1) OR (TRANSMIT$INT=0)

SEND;

SEND=$ASCII$KEY;

TRANSMIT$INT=0;

END;

ELSE

DO;

SEND=$ASCII$KEY;

FIFO=FIFO+1;

SEND=$ASCII$KEY;

TRANSMIT$INT=1;

END;

END TRANSMIT;

/* Procedure Auto$Repeat: This procedure will perform an auto repeat function
after a fixed delay period */

AUTO$Repeat:  
PROCEDURE;

IF NEW$KEY=1 THEN

DO;

TRANSMIT$TOGGLE=0;

TRANSMIT$COUNT=00H;

CALL TRANSMIT;  
/* First character */

NEW$KEY=0;

END;

ELSE

DO;

IF TRANSMIT$COUNT <> 00H THEN

DO;

TRANSMIT$COUNT=TRANSMIT$COUNT+1;

IF TRANSMIT$COUNT=OFFH THEN  /* Delay between first character and the second */

DO;

CALL TRANSMIT;  
/* Second character */

END;

END;

ELSE

DO;

CUR$SER$=1;

CUR$SER$COUNT=0;

IF TRANSMIT$TOGGLE=1 THEN  /* 2 Vertex frames between 3rd to Nth character */

CALL TRANSMIT;

/* 3rd through Nth character */

TRANSMIT$TOGGLE= NOT TRANSMIT$TOGGLE;

END;

END;

END AUTO$Repeat;
AP-223

PL/M-51 COMPILER CRT CONTROLLER

SELECT

/**************************** START MAIN PROGRAM ******************************/
/* BEGIN BY PUTTING ASCII CODE FOR BLANK IN THE DISPLAY RAM; */

363 1 INIT:
364 2 DO L=0 TO 7FH;
365 2 DISPLAY$RAM(L)=20H;
366 2 END;
/* INITIALIZE POINTERS, RAM BITS, ETC. */

366 1 ESC$SEQ=0;
367 1 SCAN$INT=0;
368 1 SERIALS$INT=0;
369 1 FIFO=0;
370 1 CURSERS$COUNT=0;
371 1 LLC=0;
372 1 DATATERMINALS$READY=1;
373 1 TOC=05H;
374 1 LINE=1800H;
375 1 RASTER=1800H;
376 1 DISPLAY$RAM$POINTER=0000H;
377 1 TRANSMIT$INT=1;

$IF SW1
378 2 DO I=0 TO 7;
379 2 LAST$KEY(I)=00H;
380 2 END;

381 1 VALID$KEY=0;
382 1 LAST$SHIFT$KEY=1;
383 1 LAST$CONTROL$KEY=1;
384 1 LAST$CAPS$LOCK=1;
SENDIF

$IF SW2
385 1 RCV$FLG=0;
386 1 SYND=0;
387 1 HYFIN=0;
388 1 KBOINT=0;
389 1 ERROR=0;
SENDIF

/* INITIALIZE THE 8276 */

385 1 COMMANDS$ADDRESS=00H;  /* RESET THE 8276 */
386 1 PARAMETERS$ADDRESS=4FH;  /* NORMAL ROWS, 80 CHARACTER/ROW */
387 1 PARAMETERS$ADDRESS=58H;  /* 2 ROW COUNTS PER VERTICAL RETRACE
25 ROWS PER FRAME */
388 1 PARAMETERS$ADDRESS=89H;  /* LINE 9 IS THE UNDERLINE POSITION
10 LINES PER ROW */
389 1 PARAMETERS$ADDRESS=0F9H;  /* OFFSET LINE COUNTER, NON-TRANSPARENT FIELD ATTRIBUTE

10-119
PL/M-51 COMPILER CRT CONTROLLER

NON-BLINKING UNDERLINE CURSORS, 20 CHARACTER COUNTS PER HORIZONTAL RETRACE */

390 1 TEMP=COMMAND$ADDRESS;
391 1 CURSOR$COLUMN=00H;
392 1 CURSOR$ROW=00H;
393 1 CURSOR$COLUMN=00H;
394 1 CALL LOADCURSERS;
395 1 TEMP=COMMAND$ADDRESS;
396 1 COMMAND$ADDRESS=050H; /* PRESET 8276 COUNTERS */
397 1 TEMP=COMMAND$ADDRESS;
398 1 COMMAND$ADDRESS=23H; /* START DISPLAY */
399 1 COMMAND$ADDRESS=00H; /* ENABLE INTERRUPTS */
400 1 TEMP=COMMAND$ADDRESS;

/* SET UP INTERRUPTS AND PRIORITIES */

401 1 $IF SW1
402 1 IP=10H; /* SERIAL PORT HAS HIGHEST PRIORITY */
403 1 ID=85H; /* ENABLE 8051 EXTERNAL INTERRUPTS */
404 1 SENDIF

405 1 $IF SW2
406 1 IP=10H; /* SERIAL PORT HAS HIGHEST PRIORITY */
407 1 ID=87H; /* ENABLE TIMER0 INTERRUPTS */
408 1 TMOD=05H; /* TIMER 0 EVENT COUNTER */
409 1 TLO=OFFH;
410 1 TH0=OFFH;
411 1 TR0=1;
412 1 $ENDIF

/* PROCEDURE SCANNER: THIS PROCEDURE SCANS THE KEYBOARD AND DETERMINES IF A SINGLE VALID KEY HAS BEEN PUSHED. IF TRUE THEN THE ASCII EQUIVALENT WILL BE TRANSMITTED TO THE HOST COMPUTER. */

413 1 SCANNER:
414 1 EA=1;
415 1 DATA$TERMINALS$READY=0;
416 1 IF CURSOR$COUNT=1FH THEN /* PROGRAMMABLE CURSOR BLINK */
417 2 DO;
418 2 CURSOR$COUNT=CURSOR$COUNT;
419 2 CURSOR$COUNT=00H;
420 2 IF CURSOR$COUNT=0 THEN
421 3 CURSOR$COUNT=7FH;
422 3 CALL LOADCURSERS;
423 3 END;
424 2 ELSE
425 2 CALL CHECK$BAUD$RATE;
426 2 LLC=LOCAL$LINE;
427 2 END;
428 2 $IF SW1
429 2 DO WHILE SCAN$INT=0; /* WAIT UNTIL VERTICAL RETRACE BEFORE */
430 3 IF SERIAL$INT=1 THEN /* SCANNING THE KEYBOARD */
431 4 CALL DECIPHER;
432 4 END;
433 3 END;
434 2 END;
CALL READER;
IF VALISEKYE =1 AND SAME =1 AND (LAST$SHIFTSKEY=SHIFTSKEY) AND (LAST$CAPSLOCK=CAPSLOCK) AND (LAST$CONTROLKEY=CONTROLKEY) THEN
CALL AUTO$REPEAT;
ELSE
DO;
IF KEY =0 AND SAME =0 THEN
DO;
TEMP =0;
K =0;
DO WHILE LAST$KEY (K) =0;
K =K+1;
END;
TEMP=LAST$KEY (K);
DO I=(K+1) TO 7;
TEMP =TEMP LAST$KEY (I);
END;
IF TEMP =LAST$KEY (K) THEN
DO;
J =0;
DO WHILE (TEMP AND 01H) =0;
TEMP =SHR (TEMP,1); J=J+1;
END;
IF TEMP >1 THEN
DO;
VALID$KEY =0;
NEWSKEY =0;
END;
ELSE
DO;
IF CONTROL$KEY =0 THEN
ASCII$KEY = (LOWSCAN (K).KEY (J)) AND 1PH;
ELSE
DO;
IF SHIFTSKEY =0 THEN
ASCII$KEY = (LOWSCAN (K+O8H).KEY (J));
ELSE
DO;
ASCII$KEY = (LOWSCAN (K).KEY (J));
IF (CAPSLOCK =0) AND (ASCII$KEY >60H) AND (ASCII$KEY <7BH) THEN
ASCII$KEY = ASCII$KEY -20H;
IF LLC =0 THEN
DO;
IF ASCII$KEY =1BH THEN
ESC$SEQ =1;
ELSE
ESC$SEQ =0;
END;
END;
END;
END;
LIST$SHIFTSKEY = SHIFTSKEY;
LAST$CAPSLOCK = CAPSLOCK;
LAST$CONTROLKEY = CONTROLKEY;
VALID$KEY =1;
NEWSKEY =1;
END;
ELSE
DO;
VALID$KEY =0;
NEWSKEY =0;
END;
END;
END;
END;
END;
END;$ENDIF
PL/M-51 COMPILER

SUBJECT

SIP SW2
IF SERIAL.INT = 1 THEN
  CALL DECIIPHER;
  IF KB.INT = 1 THEN
    DO;
      IF ERROR = 0 THEN
        DO;
          ASCII$KEY = LST$KEY (1);
          NEWSKEY = 1;
          CALL AUTO$REPEAT;
          KB.INT = 0;
        END;
        ERROR = 0;
        KB.INT = 0;
      ENIO;
    END;
    GOTO SCANNER;
  END;
END;

485 1 GOTO SCANNER;
486 1 END;

MODULE INFORMATION: (STATIC+OVERLAYABLE)
   CODE SIZE = 08E6H  2278D
   CONSTANT SIZE = 0980H  128D
   DIRECT VARIABLE SIZE = 20H+00H  45D+  0D
   INDIRECT VARIABLE SIZE = 00H+00H  0D+  0D
   BIT SIZE = 10H+00H  16D+  0D
   BIT-ADDRESSABLE SIZE = 00H+00H  0D+  0D
   AUXILIARY VARIABLE SIZE = 0000H  0D
   MAXIMUM STACK SIZE = 000CH  12D
   REGISTER-BANK(S) USED:
   1056 LINES READ
   0 PROGRAM ERROR(S)

END OF PL/M-51 COMPILATION
AP-223

MCS-51 MALRU ASSEMBLY

IIS-I1 MCS-51 MALRU ASSEMBLY V2.1
OBJECT MODULE PLACED IN IIIPIC:MTASM.UBJ
ASSEMPLER INVOKED BY: AS/51 IIIPIC:MTASM.SHC

LOC  UBJ  LINE  SOURCE

1
2
3
4 1
5 1
6
7  PUBLIC BLANK
8  PUBLIC BLINE
9  PUBLIC FILL
10 EXTRN DATA (LINEU,MASTER,POINT,SERIAL,FIFO,CURSEQ,COUNT,L)
11 EXTRN BIT (SEMIH,ESCSEQ,THNINT,SCAN)
12
13
14 CSEG AT(USH)  ;RESET RASTER TO LINEO AND SCAN KEYBOARD
15
16
17 1  EXTRN CODE (UETALM)
18 1 CSEG AT(VBM)
19 1 LJMP UETALM  ;NEEDED IF DECODED KEYBOARD IS USED
20
21 CSEG AT(U13M)
22 SJMP BUFFER  ;FILL 8276 RWM BUFFER
23
24 CSEG AT(U23M)
25 SJMP SERBUF  ;STICK SERIAL INFORMATION INTO THE FIFU
26
27 CSEG
28
29 VENT: PUSH PSR  ;PUSH NEW USED BY PLM51
30
31
32  MOV MASTER,LINEO  ;REINITIALIZE MASTER TU LINEU
33  MOV MASTER+1,LINEO+1
34  MOV NO,#01H  ;CLR 8276 INTERRUPT FLAG
35  MOVX A,#RU
36  INC COUNT  ;INC CURSOR COUNT REGISTER
37  SETB SCAN  ;FOR DEBUGGING ROUTINE
38  POP W0H  ;POP REGISTERS
39  POP ACH
40  POP UPH
41  POP PSR
42
43  BUFFER: PUSH PSR  ;PUSH ALL NEW USED BY PLM51 CODE
44  PUSH ACH
45  PUSH UPL
46  PUSH UPH
47  PUSH UPM
48  ACALL UMA  ;FILL 8276 RWM BUFFER
49  POP UPH  ;POP REGISTERS
50  POP UPL
51  POP ACH
52  POP PSR
53  HEU  ;SELECT
53  +1  SELECT
54
55
10-123
**MC6800 HALRU ASSEMBLER CRTASM**

**SYMBOL TABLE LISTING**

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>D AUCH</td>
<td>0VEUH</td>
<td>A</td>
</tr>
<tr>
<td>BLANK</td>
<td>C AUCH</td>
<td>0V85H</td>
<td>A PUB</td>
</tr>
<tr>
<td>BLINK</td>
<td>C AUCH</td>
<td>0V49H</td>
<td>A PUB</td>
</tr>
<tr>
<td>BUFFER</td>
<td>C AUCH</td>
<td>0V3FH</td>
<td>A</td>
</tr>
<tr>
<td>CHECK</td>
<td>C AUCH</td>
<td>01A3H</td>
<td>A</td>
</tr>
<tr>
<td>CUN1</td>
<td>C AUCH</td>
<td>0U06H</td>
<td>A</td>
</tr>
<tr>
<td>CUN2</td>
<td>C AUCH</td>
<td>0VE7H</td>
<td>A</td>
</tr>
<tr>
<td>COUNT</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>CURSE</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>DOUNE</td>
<td>C AUCH</td>
<td>0UF8H</td>
<td>A</td>
</tr>
<tr>
<td>DMA</td>
<td>C AUCH</td>
<td>0VFAM</td>
<td>A</td>
</tr>
<tr>
<td>DMADV</td>
<td>C AUCH</td>
<td>0186H</td>
<td>A</td>
</tr>
<tr>
<td>DONE</td>
<td>C AUCH</td>
<td>01B4H</td>
<td>A</td>
</tr>
<tr>
<td>DPH</td>
<td>D AUCH</td>
<td>0V83H</td>
<td>A</td>
</tr>
<tr>
<td>DPL</td>
<td>D AUCH</td>
<td>0V8CH</td>
<td>A</td>
</tr>
<tr>
<td>EIGHTY</td>
<td>C AUCH</td>
<td>01A1H</td>
<td>A</td>
</tr>
<tr>
<td>ESCSW</td>
<td>B AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>FIFU</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>FIFTY</td>
<td>C AUCH</td>
<td>01F3H</td>
<td>A</td>
</tr>
<tr>
<td>FILL</td>
<td>C AUCH</td>
<td>00D0H</td>
<td>A PUB</td>
</tr>
<tr>
<td>FURTY</td>
<td>C AUCH</td>
<td>0151H</td>
<td>A</td>
</tr>
<tr>
<td>GBACK</td>
<td>C AUCH</td>
<td>0U84H</td>
<td>A</td>
</tr>
<tr>
<td>L</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>LINEX</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>NUTYET</td>
<td>C AUCH</td>
<td>0V97H</td>
<td>A</td>
</tr>
<tr>
<td>OVEH</td>
<td>C AUCH</td>
<td>0V59H</td>
<td>A</td>
</tr>
<tr>
<td>OVENI</td>
<td>C AUCH</td>
<td>0U79H</td>
<td>A</td>
</tr>
<tr>
<td>POINT</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>PSW</td>
<td>D AUCH</td>
<td>00DUH</td>
<td>A</td>
</tr>
<tr>
<td>RASTER</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>SBUF</td>
<td>D AUCH</td>
<td>0V99H</td>
<td>A</td>
</tr>
<tr>
<td>SCAN</td>
<td>B AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>SBUF</td>
<td>C AUCH</td>
<td>0U52H</td>
<td>A</td>
</tr>
<tr>
<td>SERIAL</td>
<td>D AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>SERRIN</td>
<td>B AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>SEVEN</td>
<td>B AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>SIXY</td>
<td>C AUCH</td>
<td>0179H</td>
<td>A</td>
</tr>
<tr>
<td>TEN</td>
<td>C AUCH</td>
<td>0113H</td>
<td>A</td>
</tr>
<tr>
<td>THIRTY</td>
<td>C AUCH</td>
<td>0131H</td>
<td>A</td>
</tr>
<tr>
<td>TKINI</td>
<td>B AUCH</td>
<td>-------</td>
<td>EX</td>
</tr>
<tr>
<td>TRENITY</td>
<td>C AUCH</td>
<td>0124H</td>
<td>A</td>
</tr>
<tr>
<td>VERT</td>
<td>C AUCH</td>
<td>0U25H</td>
<td>A</td>
</tr>
</tbody>
</table>

**REGISTER BANK(S) USED: U**

**ASSEMBLY COMPLETE, NO ERRORS FOUND**
MC9-51 MALRU ASSEMBLER

CMTASM

LUC OBJ LINE SOURCE

0U5C 300006 56 57 SENG, BUSI J-MW U99, VGH, CVEN IF TRANSMIT BIT AUT SET THEN CHECK RECEIVE
0U5S 2099 F 56 CM 99h CLR R8H ; CLEAR TRANSMISSION INTERRUPT FLAG
0U5F U20 F 54 SETB ITRANINT ;SET TRAN INT FOR PLMS1 STATUS CHECK
0U54 20042 6b UVERI: JB V99, UCBANK ; IF RI NOT SET GUBACK
0U5C C001 61 PUSH U1
0U5E A999 62 MOV R1, 80h ; READ SBUF
0U6U C288 F 63 CLF UPL ; CLEAR RH ; PUSH REGISTERS USED BY PLMS1.
0U62 C000 66 PUSH PSH
0U64 C000 65 PUSH ACC
0U66 C000 66 PUSH 00m
0U68 E100 F 67 CLK ESC8SEG ; CLR ESC SEQUENCE FLAG
0U6A 7400 F 68 MOV A, #SERIAL ; GET SERIAL FIFO RAM START LOCATION
0U6C 7500 F 69 ADD A, #FIFC ; AND FIND ROM FAN INTO THE FIFO WE ARE
0U6E F8 F 70 MOV #O, A ; PUT IT INTO RU
0U6F E9 F 71 MOV A, #1
0U70 C27 F 72 CLK U7TH ; CLR BIT 7 OF AC1
0U72 F6 F 73 MOV #0, A ; PUT DATA IN FIFO
0U73 0A802 F 74 CJNE A, #1001001, OVEH1 ; IF DATA IS NOT A ESC KEY THEN GO OVER
0U76 U200 F 75 SETB ESC8SEG ; SET ESC SEQUENCE FLAG
0U76 U500 F 76 UVERI: INC FIFC ; MOV FIFO TO NEXT LOCATION
0U7A U200 F 77 SETB SENINT ; SET SERIAL INT BIT FOR PLMS1 STATUS CHECK
0U7C U000 78 POP 00m ; POP REGISTERS
0U7E U000 79 POP ACC
0U80 U000 80 POP PSH
0U82 U01 F 81 POP 01m
0U84 32 F 82 GOBACK: RETI ; PUSH REGS USED BY PLMS1
0U85 C000 F 84 BLANKI PUSH P8h
0U87 C000 85 PUSH ACC
0U89 C082 F 86 PUSH DPL
0U8B C063 F 87 PUSH DPH
0U8U C000 F 88 PUSH 00m
0U8F 650062 F 89 MOV UPL, LINE0+1 ; GET LINE0 INFO
0U9e 650063 90 MOV UPL, LINE0 ; AND PUT IT INTO DPH
0U95 7850 91 MOV N0, #50h ; NUMBER OF CHARACTERS IN A LINE
0U97 3700 92 NOTYTEI MOV A, #52h ; ASCII SPACE CHARACTER
0U99 F0 93 MOVX #DPR1, A ; MOV TO DISPLAY RAM
0U9A 43 94 INC DOPR ; INCH TO NEXT DISPLAY RAM LOCATION
0U9b 180A 95 DJNZ N0, NOTYTEI ; IF ALL 50H LOCATIONS ARE NOT FILLED
0U9e U000 97 POP U0h
0U9F 0005 98 POP UPH
0UAF U062 99 POP DPL
0U92 U060 100 POP ACC
0U95 U000 101 POP P8h
0U97 22 102 MET 103 +1 SELECT

10-125
### MCS-51 MACRO ASSEMBLER

#### CRTASK

<table>
<thead>
<tr>
<th>LUC U8J</th>
<th>LINE SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>00A0 C0U0 104</td>
<td><strong>LINE:</strong> PUSH PSR</td>
</tr>
<tr>
<td>00A0 C0U0 105</td>
<td></td>
</tr>
<tr>
<td>00A0 C0U0 106</td>
<td>PUSH ACL</td>
</tr>
<tr>
<td>00A0 C0U0 107</td>
<td>PUSH UPL</td>
</tr>
<tr>
<td>00A0 C0U0 108</td>
<td>PUSH UPH</td>
</tr>
<tr>
<td>00A0 C0U0 109</td>
<td>PUSH V0H</td>
</tr>
<tr>
<td>00B2 5500B3 F 110</td>
<td>MOV UPPL,PCINT</td>
</tr>
<tr>
<td>00B3 5500B3 F 111</td>
<td>MOV UPPL,PCINT+1</td>
</tr>
<tr>
<td>00BB 436310 112</td>
<td>URL UPH,#10H</td>
</tr>
<tr>
<td>00Bd A600 F 113</td>
<td>MOV #0,CURSLEN</td>
</tr>
<tr>
<td>00Bd 1620 114</td>
<td>CONT1 MOV A,#20H</td>
</tr>
<tr>
<td>00Bd F0 115</td>
<td>MOVX a0,PTM.A</td>
</tr>
<tr>
<td>00C0 A3 116</td>
<td>INC UPTR</td>
</tr>
<tr>
<td>00C1 08 117</td>
<td>INC HO</td>
</tr>
<tr>
<td>00C2 5500F8 118</td>
<td>CJNE HO,#50H,CONT1</td>
</tr>
<tr>
<td>00C5 U0U0 119</td>
<td>POP V0M</td>
</tr>
<tr>
<td>00C7 U0U3 120</td>
<td>POP UPH</td>
</tr>
<tr>
<td>00C7 U0U2 121</td>
<td>POP UPL</td>
</tr>
<tr>
<td>00C8 U0U0 122</td>
<td>POP ACC</td>
</tr>
<tr>
<td>00C9 U0U0 123</td>
<td>POP PSR</td>
</tr>
<tr>
<td>00CF 22 124</td>
<td>RET</td>
</tr>
<tr>
<td>00D0 C0U0 125</td>
<td><strong>FILL:</strong> PUSH PSR</td>
</tr>
<tr>
<td>00D2 C0U0 126</td>
<td>PUSH ACL</td>
</tr>
<tr>
<td>00D4 C0U0 127</td>
<td>PUSH UPL</td>
</tr>
<tr>
<td>00D6 C0U3 128</td>
<td>PUSH UPH</td>
</tr>
<tr>
<td>00D8 C0U0 129</td>
<td>PUSH V0H</td>
</tr>
<tr>
<td>00DA C3 130</td>
<td>CLR C</td>
</tr>
<tr>
<td>00DE 5500B2 F 131</td>
<td>MOV UPPL,#L</td>
</tr>
<tr>
<td>00E1 436310 132</td>
<td>URL UPH,#10H</td>
</tr>
<tr>
<td>00E4 184F 133</td>
<td>MOV HO,#4FH</td>
</tr>
<tr>
<td>00E6 A3 134</td>
<td>INC UPTR</td>
</tr>
<tr>
<td>00E7 1420 135</td>
<td>CONT1 MOV A,#20H</td>
</tr>
<tr>
<td>00EE F0 136</td>
<td>MOVX a0,PTM.A</td>
</tr>
<tr>
<td>00FA A3 137</td>
<td>INC UPTR</td>
</tr>
<tr>
<td>00FB U0RA 138</td>
<td>DJNZ: #0,CURAT2</td>
</tr>
<tr>
<td>00FD 00U0 139</td>
<td>POP V0M</td>
</tr>
<tr>
<td>00FE 00U3 140</td>
<td>POP UPH</td>
</tr>
<tr>
<td>00F1 U0U2 141</td>
<td>POP UPL</td>
</tr>
<tr>
<td>00F5 U0U0 142</td>
<td>POP ACC</td>
</tr>
<tr>
<td>00F7 22 143</td>
<td>RET</td>
</tr>
<tr>
<td>150</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td></td>
</tr>
<tr>
<td>152 +1 JREJLT</td>
<td></td>
</tr>
</tbody>
</table>
; THIS ROUTINE MOVES DISPLAY RAM DATA TO MON BUFFER OF 8276

00FAeldon

00FBeldon

00FCeldon

00FDeldon

00FEeldon

00FFeldon
<table>
<thead>
<tr>
<th>LUC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>013U A3</td>
<td>008</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0131 A0</td>
<td>009</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0132 A3</td>
<td>010</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0133 A0</td>
<td>011</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0134 A3</td>
<td>012</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0135 A0</td>
<td>013</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0136 A3</td>
<td>014</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0137 A0</td>
<td>015</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0138 A3</td>
<td>016</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0139 A0</td>
<td>017</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>013A A3</td>
<td>018</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>013B A0</td>
<td>019</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>013C A3</td>
<td>020</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>013D A0</td>
<td>021</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>013E A3</td>
<td>022</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>013F A0</td>
<td>023</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0140 A3</td>
<td>024</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0141 A0</td>
<td>025</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0142 A3</td>
<td>026</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0143 A0</td>
<td>027</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0144 A3</td>
<td>028</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0145 A0</td>
<td>029</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0146 A3</td>
<td>030</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0147 A0</td>
<td>031</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0148 A3</td>
<td>032</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0149 A0</td>
<td>033</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>014A A3</td>
<td>034</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>014B A0</td>
<td>035</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>014C A3</td>
<td>036</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>014D A0</td>
<td>037</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>014E A3</td>
<td>038</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>014F A0</td>
<td>039</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0150 A3</td>
<td>040</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0151 A0</td>
<td>041</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0152 A3</td>
<td>042</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0153 A0</td>
<td>043</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0154 A3</td>
<td>044</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0155 A0</td>
<td>045</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0156 A3</td>
<td>046</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0157 A0</td>
<td>047</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0158 A3</td>
<td>048</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0159 A0</td>
<td>049</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>015A A3</td>
<td>050</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>015B A0</td>
<td>051</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>015C A3</td>
<td>052</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>015D A0</td>
<td>053</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>015E A3</td>
<td>054</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>015F A0</td>
<td>055</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0160 A3</td>
<td>056</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0161 A0</td>
<td>057</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0162 A3</td>
<td>058</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0163 A0</td>
<td>059</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0164 A3</td>
<td>060</td>
<td>INC UPIR</td>
<td></td>
</tr>
<tr>
<td>0165 A0</td>
<td>061</td>
<td>MOVX A, #DPK</td>
<td></td>
</tr>
<tr>
<td>0166 A3</td>
<td>062</td>
<td>INC UPIR</td>
<td></td>
</tr>
</tbody>
</table>

<p>| 10-128 |</p>
<table>
<thead>
<tr>
<th>LUC UPJ</th>
<th>LNE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>016/ E0</td>
<td>c63</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c64</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c65</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c66</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c67</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c68</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c69</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c70</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c71</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c72</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c73</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c74</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c75</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c76</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c77</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c78</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>c79</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>c80</td>
<td>INCI UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b81</td>
<td>SIATYI MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b82</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b83</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b84</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b85</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b86</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b87</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b88</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b89</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b90</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b91</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b92</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b93</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b94</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b95</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b96</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b97</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b98</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b99</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b100</td>
<td>INC UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b101</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b102</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b103</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b104</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b105</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b106</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b107</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b108</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b109</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b110</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b111</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b112</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b113</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b114</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b115</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>016/ A3</td>
<td>b116</td>
<td>INCl UP1R</td>
</tr>
<tr>
<td>016/ E0</td>
<td>b117</td>
<td>MOVX A,A0RTH</td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>line</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>019E A3</td>
<td>318</td>
<td>INC</td>
</tr>
<tr>
<td>019F E0</td>
<td>319</td>
<td>MOVX</td>
</tr>
<tr>
<td>01A0 A3</td>
<td>320</td>
<td>INC</td>
</tr>
<tr>
<td>01A1 E0</td>
<td>321</td>
<td>EIGHTY</td>
</tr>
<tr>
<td>01A2 A3</td>
<td>322</td>
<td>INC</td>
</tr>
<tr>
<td>01A3 E563</td>
<td>323</td>
<td>CHECK</td>
</tr>
<tr>
<td>01A5 #1F0C</td>
<td>324</td>
<td>CJNE</td>
</tr>
<tr>
<td>01A6 E562</td>
<td>325</td>
<td>MOV</td>
</tr>
<tr>
<td>01A7 #0007</td>
<td>326</td>
<td>CJNE</td>
</tr>
<tr>
<td>01A8 750000 F</td>
<td>327</td>
<td>MOV</td>
</tr>
<tr>
<td>01B6 750000 F</td>
<td>328</td>
<td>MOV</td>
</tr>
<tr>
<td>01B7 553U00</td>
<td>329</td>
<td>MET</td>
</tr>
<tr>
<td>01B8 E56C</td>
<td>330</td>
<td>DONE</td>
</tr>
<tr>
<td>01B9 E56O</td>
<td>331</td>
<td>MET</td>
</tr>
<tr>
<td>01BC E562</td>
<td>332</td>
<td>UMADE</td>
</tr>
<tr>
<td>01BD E44F</td>
<td>333</td>
<td>MOV</td>
</tr>
<tr>
<td>01C7 5582</td>
<td>334</td>
<td>AUX</td>
</tr>
<tr>
<td>01C5 S00F</td>
<td>335</td>
<td>MOV</td>
</tr>
<tr>
<td>01C6 5583</td>
<td>336</td>
<td>JNC</td>
</tr>
<tr>
<td>01C7 60UB</td>
<td>337</td>
<td>INC</td>
</tr>
<tr>
<td>01C8 60UB</td>
<td>338</td>
<td>SJMP</td>
</tr>
<tr>
<td>343</td>
<td>344</td>
<td>345</td>
</tr>
</tbody>
</table>

10-130
AP-223

MCS-51 HALRU ASSEMBLER KEYBD

I3IB-II MCS-51 HALRU ASSEMBLER v2.1
OBJECT MODUFE PLACED IN: KEYBD.OBJ
ASSEMBLER INVOKED AT: A551 IIFKEYBU.SRC

LOC OBJ LINE SOURCE

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
; ; ; ;
THIS CURRSTNS THE SOTFMAKE NEEDED TO SCAR AN UNDECENDED KEYBOARD
THIS PROGRAM MUST BE LINKED TO THE MAIN PROGRAMS TO FUNCTION
MEMORY MAP FOR READINT KEY BOARD (USING KVC)
ADDRESS PCK KEY BOARD 10FFH TO 17FFH
PUBLIC READEN
EXTERN DATA (LTKEY)
EXTERN BIT (KEYUJAME)
PUBLIC READER
"READEN"ROUTINE
SEECT
MLB-51 HALFW-ASSEMBLER KEYBO

LOC OBJ LINE SOURCE

3/ 3. UNDECoded KEYBOARD SEGMENT CUDL
3Y MSEG UNDECoded KEYBOARD
40
41
42

0000 COO0 42 HEADER: PUSH PSH ;PUSH NEW USED BY PLMS1
0002 CO00 44 PUSH ACL
0004 CO02 45 PUSH DFL
0006 CO03 46 PUSH UPN
0008 CO04 47 PUSH UOH
000A CO01 48 PUSH U1m
000C CO02 49 PUSH U2m
000E CO03 50 PUSH U3m
0100 10FF 51 MOV UPTR,#10FFh ;INITIALIZE UPTR TO KEYBOARD
52
0113 190F 53 MOV M1,#00m ;CLR ZERO COUNTER
0115 1808 F 54 MOV M0,#00m ;GET KEYBOARD RAM POINTER
0117 1808 55 MOV M3,#00m ;INITIALIZE LOOP COUNTER
0119 L200 F 56 CLR KEYO ;INITIALIZE PLMS1 STATUS BITS
011B 0200 F 57 SETB SAME
011D 0602 58 MOV M2h,#00 ;MOV LAST KEYBOARD SCAN TO 02h
011F E4 59 CLK: A
0120 09 60 MOV C,A,#A+DPTH ;SCAN KEYBOARD
0121 F4 61 CPL A ;INVERT
0122 6005 62 JZ ZEHC ;IF SCAN WAS ZER0 GO INCREMENT ZER0 COUNTER
0124 55224 63 CJNE A,#02h,NTSAME ;COMPARE WITH LAST SCAN IF NOT THE SAME
0126 66 64 THEN CLR SAME BIT AND WRITE NEW INFORMATION
0127 005 65 TO RAM
0129 0501 66 DJNO: INC Q1m1 ;INCH ZERO COUNTER
012B 05210 67 CJNE A,#02h,NTSAME ;STEP TO NEXT SCAN RAM LOCATION
012D 08 68 EQUAL: INC W0 ;STEP TO NEXT SCAN RAM LOCATION
012F 0503 69 INC UPN ;NEXT KEYBOARD ADDRESS
0131 LRLA 71 DJNZ M3,#PURF ;IF LOOP COUNTER NOT 0, SCAN AGAIN
0133 5900 72 CJNE M1,#00m,BACK ;CHECK TO SET IF ALL 8 SCANS WHERE 0
0135 6100 73 SETB KEYO ;IF YES SET KEYO BIT
0137 C200 F 74 CLK: SAME
0139 0003 F 75 POP U3m
013B 0002 76 POP C2h
013D 0001 77 POP 01m
0140 0000 78 POP 00m
0142 0003 79 POP UPN
0144 0002 80 POP DFL
0146 0000 81 POP ACL
0148 0000 82 POP PSR
014A 4283 83 RET
014B F6 85 NTSAME: MOV A,R0,A ;IF SCAN WAS NOT THE SAME THEN PUT NEW
014D 2D00 F 87 CLK: SAME ;SCAN INPC INTO RAM
014F 800E 88 SJMP EQUAL ;CLR SAME BIT
0150 94 89
0151 ENU
<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALC</td>
<td>D</td>
<td>00E4H</td>
<td>A</td>
</tr>
<tr>
<td>BACK</td>
<td>C</td>
<td>003AH</td>
<td>R</td>
</tr>
<tr>
<td>D'W</td>
<td>D</td>
<td>0038H</td>
<td></td>
</tr>
<tr>
<td>DPL</td>
<td>D</td>
<td>0082H</td>
<td>A</td>
</tr>
<tr>
<td>EJUAI</td>
<td>C</td>
<td>002EH</td>
<td>R</td>
</tr>
<tr>
<td>KEYU</td>
<td>B</td>
<td>--------</td>
<td>EXT</td>
</tr>
<tr>
<td>LSTKEY</td>
<td>D</td>
<td>--------</td>
<td>EXT</td>
</tr>
<tr>
<td>MURE</td>
<td>C</td>
<td>0010H</td>
<td>R</td>
</tr>
<tr>
<td>NISAML</td>
<td>C</td>
<td>004BH</td>
<td>R</td>
</tr>
<tr>
<td>PBA</td>
<td>C</td>
<td>00D8H</td>
<td></td>
</tr>
<tr>
<td>READER</td>
<td>C</td>
<td>0000H</td>
<td>R</td>
</tr>
<tr>
<td>SANE</td>
<td>E</td>
<td>--------</td>
<td>EXT</td>
</tr>
<tr>
<td>UNDECoded_Keyboard</td>
<td>SEG</td>
<td>0059H</td>
<td>R</td>
</tr>
<tr>
<td>ZERO</td>
<td>C</td>
<td>0029H</td>
<td>SEG=UNDECoded_Keyboard</td>
</tr>
</tbody>
</table>

REGISTER BANK(S) USEG: U

ASSEMBLY COMPLETE, NO ERRORS FOUND
MCS-51 MACRO ASSEMBLER

DECIDE

IBID-II MCS-51 MACRO ASSEMBLER V2.1
OBJECT MODULE PLACED IN IFI\DECIDE.UBJ
ASSEMBLER INVOKED BY ASP51 IFI\DECIDE.SHC

LOC UBJ LINE SOURCE
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26 +1

PUBLIC DETACH
EXTRN DATA (L81KEY)
EXTRN CIT (KBUNIT)

/* "DECIDE" INTERRUPT ROUTINE FOR DECODED KEYBOARDS */

//EJECT
MLS-51 MACRO ASSEMBLER

DECIDE

LOC UBJ LINE SOURCE

27 DECODED_KEYBOARD SEGMENT CODE
29 SEG DECODED_KEYBOARD

0160 LOAD 31 DESTACK PUSH PHR
0162 LOAD 32 PUSH PLR
0164 LOAD 33 PUSH PHP
0166 LOAD 34 PUSH ACC
0168 LOAD 35 MOV UPTR,#OFF F
016A LOAD 36 CLR A
016C LOAD 37 MOVCA A,#DPH
016E LOAD 38 MOV LSIKEY1,A
0170 LOAD 39 SETB RBINT
0172 LOAD 40 MOV UP,#OFF F
0174 LOAD 41 MOV ILU,#OFF F
0176 LOAD 42 POP ACC
0178 LOAD 43 POP DPH
017A LOAD 44 POP PLR
017C LOAD 45 POP PHP
017E LOAD 46 RETI

51 END
**MCS-51 MACRO ASSEMBLER**

**SYMBOL TABLE LISTING**

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>D</td>
<td>0UE0H</td>
<td>A</td>
</tr>
<tr>
<td>DECoded_KEYBOARD</td>
<td>C</td>
<td>0U20H</td>
<td>REL=Unit</td>
</tr>
<tr>
<td>DETACH</td>
<td>C</td>
<td>0U00H</td>
<td>PUB SEG=DECoded_KEYBOARD</td>
</tr>
<tr>
<td>DPH</td>
<td>D</td>
<td>0U83H</td>
<td>A</td>
</tr>
<tr>
<td>DPL</td>
<td>D</td>
<td>0U82H</td>
<td>A</td>
</tr>
<tr>
<td>DnDINT</td>
<td>E</td>
<td>----</td>
<td>EXT</td>
</tr>
<tr>
<td>LStKEY</td>
<td>D</td>
<td>----</td>
<td>EXT</td>
</tr>
<tr>
<td>PSW</td>
<td>D</td>
<td>0U00H</td>
<td>A</td>
</tr>
<tr>
<td>THO</td>
<td>D</td>
<td>0U8CH</td>
<td>A</td>
</tr>
<tr>
<td>TLO</td>
<td>D</td>
<td>0U8AH</td>
<td>A</td>
</tr>
</tbody>
</table>

**REGISTRY BANK(S) USED: 0**

**ASSEMBLY COMPLETE, NO ERRORS FOUND**
AP-223

ML3-51 HALRU ASSEMBLY DETACH

ISIS-11 ML3-51 HALRU ASSEMBLY V2.1
OBJECT MODULE PLACED IN 1F1DETACH.UBJ
ASSEMBLER INVOKED BY: ASR51 1F1DETACH.SNC

LUC UBJ LIME SOURCE

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

;********************************************************
;*******************************************************************************************
;******************************************************************************
;******************************************************************************
;******************************************************************************
;
;**** SOFTWARE FOR A SERIAL OR DETACHABLE KEYBOARD
;
;*******************************************************************************************
;******************************************************************************
;******************************************************************************
;******************************************************************************
;
;*******************************************************************************************
;******************************************************************************
;******************************************************************************

; THIS CONTAINS THE SOFTWARE NEEDED TO PERFORM A SOFTWARE SERIAL PORT FOR SERIAL KEYBOARDS AND DETACHABLE KEYBOARD. THIS PROGRAM MUST BE LINKED TO THE MAIN PROGRAMS FOR USE.

10 ;
11 ;
12 ;
13 ;
14 ;
15 ;
16 ;
17 ;
18 ;
19 +1 SEJECT
MC8-51 HALU ASSEMBLER

DETACh

LUC  W4J  LINE  SOURCE

20 ;
21 ;
22 ;
23 ;
24 ;
25 INPUT ECU TV
26;
27 PUBLIC DETACH
28 EXTRN U41 (LS1KEY)
29 EXTRN U11 (RCVFLU,SYNCSYF1N)
30 EXTRN BIT (KEDEM1,ERROR)
31 ;
32 ;
33 ;
34 ;
35 ;
36 ;
37 ;
38 ;
39 ;
40 ;
41 ;
42 ;
43 ;
44 ;
45 ;
46 ;
47 START0 EQU U040H 1LOW BYTE FOR 150 BAUD
48 START1 EQU U044H 1HIGH BYTE FOR 150 BAUD
49 MESSAGE0 EQU U040H 1LOW BYTE FOR 150 BAUD
50 MESSAGE1 EQU U044H 1HIGH BYTE FOR 150 BAUD
51 +1 SEJECT

TIMES LOAD VALUES FOR DIFFERENT BAUD RATES
USED WITH DETAChABLE KEYBOARDS

BAUD START BIT DETECT MESSAGE DETECT
110 UEP46H ODF45H
150 0F400H OSE00H

10-138
Detach

Detachable Keyboard Segment Code

;Push registers used by PLM51
;Push registers used by PLM51

;If receive flag set get next bit
;If receive flag set get next bit

;If to is a 1 then not a start bit
;If to is 0 then it's a start bit

;Set Timer to Interrupt in the middle of Start Bit
;Set Timer to Interrupt in the middle of Start Bit

;Set Timer Counter to time end mode
;Set Timer Counter to time end mode

;Go back to program
;Go back to program

;Check if Valid Start Bit has been seen
;Check if Valid Start Bit has been seen

;If not check if Valid Start Bit
;If not check if Valid Start Bit

;If yes set sync
;If yes set sync

;Init LSKEY
;Init LSKEY

;Set Timer for 1 bit time
;Set Timer for 1 bit time

;AND go back to main program
;AND go back to main program

;Set Timer for 1 bit time
;Set Timer for 1 bit time

;Check to see if all 8 bits have been received
;Check to see if all 8 bits have been received

;Get Working Register
;Get Working Register

;Get Next Bit from T1
;Get Next Bit from T1

;If no carry then not done
;If no carry then not done

;CLR bit 7
;CLR bit 7

;MOV final code to LSKEY+1
;MOV final code to LSKEY+1

;FINISH
;FINISH

END
<table>
<thead>
<tr>
<th>LOC</th>
<th>ORJ</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>04A</td>
<td>30d45</td>
<td></td>
<td>94</td>
<td>;</td>
</tr>
<tr>
<td>04D</td>
<td>220</td>
<td>F</td>
<td>100</td>
<td>STOP; JNB JAPLI, ERN</td>
</tr>
<tr>
<td>04F</td>
<td>22000</td>
<td>F</td>
<td>101</td>
<td>SETB KBINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>102</td>
<td>JMP KBST</td>
</tr>
<tr>
<td>05A</td>
<td>020</td>
<td>F</td>
<td>103</td>
<td>;</td>
</tr>
<tr>
<td>05B</td>
<td>220</td>
<td>F</td>
<td>104</td>
<td>ERN1 SETB ERNCr</td>
</tr>
<tr>
<td>05C</td>
<td>220</td>
<td>F</td>
<td>105</td>
<td>KSI CLR KCVFLG</td>
</tr>
<tr>
<td>05D</td>
<td>220</td>
<td>F</td>
<td>106</td>
<td>CLK Sync</td>
</tr>
<tr>
<td>06A</td>
<td>059</td>
<td>F</td>
<td>107</td>
<td>;</td>
</tr>
<tr>
<td>06B</td>
<td>222</td>
<td>F</td>
<td>108</td>
<td>MOV A, DPL</td>
</tr>
<tr>
<td>06C</td>
<td>559</td>
<td>F</td>
<td>109</td>
<td>MOV HLC, A</td>
</tr>
<tr>
<td>06D</td>
<td>75C0F</td>
<td>110</td>
<td>MOV 0V, #OFFM</td>
<td></td>
</tr>
<tr>
<td>06E</td>
<td>7500F</td>
<td>111</td>
<td>MOV 0V, #OFFM</td>
<td></td>
</tr>
<tr>
<td>06F</td>
<td>00D</td>
<td>112</td>
<td>MOV 0V, #OFFM</td>
<td></td>
</tr>
</tbody>
</table>

1. IF NOT 1 THEN NOT A VALID STOP BIT
2. TELL PLP A BYTE IS READY
3. AND GO BACK TO MAIN PROGRAM
4. CLEAR FLAGS
5. SET TIMER 0 TO COUNT INCREASE
6. SET COUNTER TO FFFF SO INTERRUPT
7. ON NEXT FALLING EDGE OF TV
### SYMBOL TABLE LISTING

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>C ADUR</td>
<td>0060H</td>
<td>A</td>
</tr>
<tr>
<td>BYFIK</td>
<td>b ADUR</td>
<td>------</td>
<td><strong>EXTERNAL</strong></td>
</tr>
<tr>
<td>DETACH</td>
<td>C ADUR</td>
<td>0080H</td>
<td><strong>SEGUE</strong></td>
</tr>
<tr>
<td>DETACHABLE_KEYBOARD</td>
<td>SEG</td>
<td>0068H</td>
<td><strong>SEL=DETACHABLE_KEYBOARD</strong></td>
</tr>
<tr>
<td>ERR</td>
<td>C ADUR</td>
<td>0052H</td>
<td><strong>HEL=UNIT</strong></td>
</tr>
<tr>
<td>ERRUR</td>
<td>b ADUR</td>
<td>------</td>
<td><strong>EXTERNAL</strong></td>
</tr>
<tr>
<td>FIN1</td>
<td>C ADUR</td>
<td>0045H</td>
<td><strong>SEG=DETACHABLE_KEYBOARD</strong></td>
</tr>
<tr>
<td>INPUT</td>
<td>b ADUR</td>
<td>0060H</td>
<td>A</td>
</tr>
<tr>
<td>KBDINT</td>
<td>b ADUR</td>
<td>------</td>
<td><strong>EXTERNAL</strong></td>
</tr>
<tr>
<td>LSTKEY</td>
<td>U ADUR</td>
<td>------</td>
<td><strong>EXTERNAL</strong></td>
</tr>
<tr>
<td>MESSAGE1</td>
<td>NLPB</td>
<td>0000H</td>
<td>A</td>
</tr>
<tr>
<td>MESSAGE2</td>
<td>NLPB</td>
<td>0068H</td>
<td>A</td>
</tr>
<tr>
<td>NABIT</td>
<td>C ADUR</td>
<td>0020H</td>
<td><strong>SEG=DETACHABLE_KEYBOARD</strong></td>
</tr>
<tr>
<td>PSM</td>
<td>U ADUR</td>
<td>0000H</td>
<td>A</td>
</tr>
<tr>
<td>RCVFLG</td>
<td>b ADUR</td>
<td>------</td>
<td><strong>EXTERNAL</strong></td>
</tr>
<tr>
<td>RST</td>
<td>C ADUR</td>
<td>0054H</td>
<td><strong>SEG=DETACHABLE_KEYBOARD</strong></td>
</tr>
<tr>
<td>START1</td>
<td>NLPB</td>
<td>0000H</td>
<td>A</td>
</tr>
<tr>
<td>START2</td>
<td>NLPB</td>
<td>00F4H</td>
<td>A</td>
</tr>
<tr>
<td>STOP1</td>
<td>C ADUR</td>
<td>004AH</td>
<td><strong>SEG=DETACHABLE_KEYBOARD</strong></td>
</tr>
<tr>
<td>SYNC</td>
<td>b ADUR</td>
<td>------</td>
<td><strong>EXTERNAL</strong></td>
</tr>
<tr>
<td>T0</td>
<td>b ADUR</td>
<td>0080H</td>
<td>A</td>
</tr>
<tr>
<td>T1</td>
<td>U ADUR</td>
<td>008CH</td>
<td>A</td>
</tr>
<tr>
<td>T2</td>
<td>U ADUR</td>
<td>008AH</td>
<td>A</td>
</tr>
<tr>
<td>T3</td>
<td>U ADUR</td>
<td>008AH</td>
<td>A</td>
</tr>
<tr>
<td>VAL1</td>
<td>C ADUR</td>
<td>001AH</td>
<td><strong>SEG=DETACHABLE_KEYBOARD</strong></td>
</tr>
</tbody>
</table>

**REGISTER BANK(S) USED:** 0

**ASSEMBLY COMPLETE, NO ERRORS FOUND**
APPENDIX B
REFERENCES

Controller chip takes on many industrial, computer uses

With industrial and computer control applications increasing all the time—and telecommunications and consumer applications emerging—designers increasingly need microcontrollers whose performance extends beyond that of the conventional 8-bit architectures. Normally, control system designers must depend on expensive and complex multiple-chip microprocessors to achieve high performance. But now, a 16-bit single-chip controller offers a much better solution. Not only does the 8096 offer perhaps the most extensive input/output “services” of any microcontroller, it also provides an instruction set and addressing modes tuned for both fast control operations and high-speed arithmetic.

In industrial applications, the 8096 can be used for process control, robotics, numerical and motor control, and instrumentation. Figure 1 shows the chip in a typical closed-loop servo system of the type used in industrial applications. In computer applications performance is the key feature, and here the 8096 provides greater throughput in systems in which simple data structures—a single I/O bit—and relatively small memories are required. Typical applications are computer peripherals such as printers, plotters, Winchester, and other hard-disk systems.

In the consumer end, moreover, the 8096 is ideally suited for automotive engine and other controls (see “Stopping a Car”) and sophisticated video games. Both applications need the speed, calculating power, and addressability of a 16-bit microcomputer. For telecommunications, the controller is intended for high-speed modems, PABXs, and central office switching systems.

In addition to the full 16-bit CPU, the 8096’s basic architecture includes an 8-kbyte ROM and a 232-byte RAM, which serves as a register file. To meet the wide needs of controller environments, the chip contains an eight-channel, 10-bit analog-to-digital converter, a full-duplex UART (universal asynchronous receiver-transmitter), two 16-bit timers, and a programmable pulse-width-modulated output.

Since a microcontroller must be able to interface with various types of transducers and sensors, the 8096 features built-in, extensive I/O facilities. These include an eight-level priority interrupt structure, full-duplex serial I/O, parallel I/O, a watchdog timer, analog inputs for the a-d converter, a pulse-width modulated output and a high-resolution pulse output. Each of these facilities is integrated not only physically but logically into the chip’s structure by being tightly coupled to the CPU.

The inherently high performance of a CPU suffers if the controller spends too much time administering complex real-time I/O operations. The 8096’s onboard I/O facilities solve this problem by permitting the CPU to devote more time to executing mathematics and control algorithms and less on I/O.

<table>
<thead>
<tr>
<th>Memory Allocation</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000-0017</td>
<td>On-chip I/O</td>
</tr>
<tr>
<td>0018-0019</td>
<td>Data register/stack pointer</td>
</tr>
<tr>
<td>001A-00FF</td>
<td>Data registers (230 bytes)</td>
</tr>
<tr>
<td>0100-1FFF</td>
<td>Off-chip expansion RAM/ROM/I/O</td>
</tr>
<tr>
<td>1FFE-1FFF</td>
<td>On-chip I/O</td>
</tr>
<tr>
<td>2000-200F</td>
<td>Internal ROM interrupt vectors</td>
</tr>
<tr>
<td>201D-207F</td>
<td>Reserved</td>
</tr>
<tr>
<td>2080-3FFF</td>
<td>Internal ROM user program space</td>
</tr>
<tr>
<td>4000-FFFF</td>
<td>Off-chip expansion RAM/ROM/I/O</td>
</tr>
</tbody>
</table>

---

Steve Wiseman, Product Marketing Manager
Steve Burton, Senior Engineer
John Katausky, Technical Marketing Manager
Intel Corp.
5000 W Williams Field Rd, Chandler, Ariz. 85224

Reprinted from ELECTRONIC DESIGN - August 5, 1982

Copyright 1982 Hayden Publishing Co., Inc.
16-bit microcontroller

The instruction set handles signed and unsigned 16-bit multiplications and divisions. Both 8-bit bytes and 16-bit double words are supported, and even 32-bit double words are supported for a subset of the main instruction set. A full 64 kbytes of memory address space is usable.

A flexible register structure

The 8096 instruction set directly supports 256 bytes of registers, which can be referenced as 128-word registers or as 64 double-word registers. These registers also appear—for memory reference instructions—as the first 256 bytes of the 64-kbyte RAM address space. This permits, for example, the use of a portion of the register space as the subroutine stack on smaller systems that do not have external expansion memory (Table 1).

The first 24 bytes of this register space are reserved for on-chip I/O addresses. I/O locations are memory-mapped and can be referenced directly as registers. The word register located at address 18H serves as the stack pointer. Such a large register space allows a programmer to keep his most frequently referenced scalar variables in registers.

---

**Table 2. Address modes of the 8096**

<table>
<thead>
<tr>
<th>Name</th>
<th>Time (μs)</th>
<th>Time (μs)</th>
<th>Written form</th>
<th>Action taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>0</td>
<td>N.A.</td>
<td>ADD B,A</td>
<td>(B) &lt;= (B) + (A)</td>
</tr>
<tr>
<td>Immediate</td>
<td>0.2 (word)</td>
<td>N.A.</td>
<td>ADD B,#A</td>
<td>(B) &lt;= (B) + A</td>
</tr>
<tr>
<td>Immediate</td>
<td>0 (byte)</td>
<td>N.A.</td>
<td>ADD B,#A</td>
<td>(B) &lt;= (B) + A</td>
</tr>
<tr>
<td>Indirect</td>
<td>0.4</td>
<td>1.4</td>
<td>ADD B,(A)</td>
<td>(B) &lt;= (B) + ((A))</td>
</tr>
<tr>
<td>Autoincrement</td>
<td>0.6</td>
<td>1.6</td>
<td>ADD B,(A)+</td>
<td>(B) &lt;= (B) + ((A)); (A) &lt;= (A) + &lt;length of A&gt;</td>
</tr>
<tr>
<td>Short indexed</td>
<td>0.4</td>
<td>1.4</td>
<td>ADD B,C[A]</td>
<td>(B) &lt;= (B) + ((A) + C); where -128 &lt; C &lt; 127</td>
</tr>
<tr>
<td>Short indexed</td>
<td>0.6</td>
<td>1.8</td>
<td>ST C[A],B</td>
<td>((A) + C) &lt;= (B) ST, Pop only</td>
</tr>
<tr>
<td>Long indexed</td>
<td>0.6</td>
<td>1.6</td>
<td>ADD B[A],C</td>
<td>(B) &lt;= (B) + ((A) + C)</td>
</tr>
<tr>
<td>Long indexed</td>
<td>0.8</td>
<td>1.8</td>
<td>ST [A]C,B</td>
<td>((A) + C) = (B) ST, Pop only</td>
</tr>
</tbody>
</table>

1. Addressed operand located in register space
2. Addressed operand located in ROM space or external memory-expansion space
Note: ST and Pop are the only instructions with an address-moded destination

---

1. Sitting at the center of a closed-loop servo system, the 8096 microcontroller's I/O facilities interface with both digital and analog input signals. The 16-bit chip can handle virtually any type of computer or industrial control-system application.
Since the number of instructions required is reduced, fewer external memory references are needed. As a result, program execution is accelerated.

The 8096 uses separate internal instruction and data buses. With this architecture, ROM and external memory references are slightly slower than register references. Instructions cannot be executed out of the internal RAM register space, but external-expansion RAM instructions can be executed.

Both memory space and register space are fully byte-addressable. A 16-bit word begins on an even byte address, and the odd byte is the most significant byte of the word. A 32-bit double word begins on an even-word address—both bit 0 and bit 1 of the address are zeros. Double words are produced by Multiply Words, Shift, and Normalize instructions and are used by Divide Words, Shift, and Normalize instructions. Double words are added and subtracted using Add with Carry and Subtract with Borrow instructions.

In most computers, the most commonly used instruction is Move. Because of the many registers in the 8096, a programmer can get away with fewer Move instructions—it is not necessary to switch

---

**Stopping a car**

The 8096 can be very useful in an automotive anti-skid braking system that allows a driver to decelerate his vehicle safely when one or more wheels start slipping. Slip conditions can be detected either by excessive wheel-speed differential or by excessive apparent deceleration, or both. For example, if one wheel hits an ice patch during heavy braking, the rotation of that wheel will slow down significantly, indicating a skid. On the other hand, normal tires begin to slip at or below 1 g of acceleration. If the apparent deceleration as measured by wheel speed is less than 1 g, the wheel is assumed to be skidding. Both skid-detection techniques depend on measuring a wheel's rotational speed.

Wheels are monitored by reluctance (magnetic) pickups, which generate pulse trains whose frequencies are proportional to wheel speed. Usually, a simple numerical relationship relates frequency to speed—13.3 Hz per miles per hour is a typical value. Four pickup outputs are easily handled by the 8096's high-speed input unit. At each transition of any of the pickups, the current value of timer 0 is saved in the input FIFO. The programmable edge detector in the high-speed input unit provides a convenient device for handling the wide dynamic range of the period measurement. At slow speeds, the edge detector can be programmed to respond to both edges of the input signal; at medium speeds, to recognize only positive-going edges; and at high speeds, to respond to just one of eight positive-going edges. This technique not only extends the dynamic range of the measurement, but also reduces the interrupt overhead at high speeds.

Three successive time samples—T_x, T_y, and T_z—allow the speed and acceleration of a wheel to be determined from the following equations:

\[ V_{xy} = C/(T_x - T_y) \]  
\[ V_{yz} = C/(T_y - T_z) \]  
\[ A_{xy} = (V_{xy} - V_{yz})/\{0.5 (T_x - T_z)\} \]

where \( C \) is the reciprocal of the proportionality constant (e.g., \( C = 1/13.3 \)). \( V_{xy} \), \( V_{yz} \), and \( V_{xy} \) are velocities, and \( A_{xy} \) is acceleration.

The period of the incoming frequency is in units of 1.6 \( \mu \)s since timer 0 is incremented at this rate. If \( I_x \) represents the value in the timer at time \( x \) and \( I_y \), the value at time \( y \), then Eq. 1 and 2 are written as:

\[ V_{xy} = 46992.48/(I_x - I_y) \]  
\[ V_{yz} = 46992.48/(I_y - I_z) \]  
\[ A_{xy} = (V_{xy} - V_{yz})/\{0.5 (T_x - T_z)\} \]

In practice, the constants in Eq. 4 and 5 should be multiplied by a scaling factor to allow calculations to be performed in integer arithmetic. A factor of 100, for example, gives speed measured in units of 1/100 of a mile per hour. Then the value 46992.48 becomes 4699248, which can easily be represented within 32 bits, and a 32-by-16-division instruction is used to perform the division.

Each wheel requires two such divisions—one for speed, one for acceleration—during each loop of the calculation. A typical loop takes about 10 ms. A typical 8-bit microprocessor takes 500 to 750 \( \mu \)s per division, which means eight such divisions would require 4 to 6 ms. But the 8096 does all eight divisions in about 50 \( \mu \)s. This speed improvement translates into a higher-performance module in response time or adaptability.

The watchdog timer of the 8096 helps enhance the reliability of the braking module. At a 15-MHz clock rate, the timer is incremented every 200 ns. During operation, the system software executes diagnostics periodically to ensure that the overall system—including hardware and software—is operating properly. If the operation is correct, the software will issue commands to reset the watchdog. But if a system failure prevents a diagnostic from running within a prescribed period, the watchdog timer will reset the entire system. The software cannot reset properly on an erroneous operation, such as a counter overflow, except by writing to the watchdog timer twice within its counting cycle.

---

11-3
16-bit microcontroller

operands in and out of memory locations. In addition, the chip’s powerful three-operand instructions—Add, Subtract, Multiply and Logical And—often eliminate them. Since programmer productivity (measured in lines of code written per day) is reasonably constant, writing fewer Move instructions can lead to reduced development expense. Register Load and Store instructions, with a full set of addressing modes, handle moves that cannot be eliminated.

Keeping addressing simple

Because a study of the ways in which addressing is used on the 8086 microprocessor indicates that programmers use complex addressing modes less than 0.7% of the time, the 8096’s instruction set bypasses those in favor of the more commonly used addressing modes. But should complex addressing be needed, programmers can build such modes through macros. Addressing modes in the 8096 include direct, register-indirect, immediate, autoincrement, and both short (8-bit) and long (16-bit) indexed-address. Table 2 lists the address modes and the operations that occur when each is activated.

Indexed-address modes, by adding an 8- or 16-bit displacement to the contents of any 16-bit register to form the effective address of an operand, allow fast access to arrays stored anywhere in memory. Indexed modes are also useful for referencing elements of based structures, as in the PL/M language. However, preliminary calculations are needed to reference a based array.

The stack pointer is fully addressable, as are all other 16-bit registers. As a result, it can be the base register for moded references. Stack-relative addressing, which is easy to program, is often used for recursive-subroutine parameter passing and dynamically allocated variables. While this technique does not make the best use of a large register space, it adds flexibility to the system. The stack need not be confined to internal RAM, but can fill any available RAM space in the system. The stack can flow across the boundary into register space at will, allowing recursion to very great subroutine depths.

Of the instruction set’s 71 instructions, 25 take on both word and byte form, which increases the total to 96 instructions. The set includes 16 varieties of conditional jump, allowing for both signed and unsigned comparisons. All of the 2048 bits in register space can be tested individually by a Jump on Bit/Not Bit instruction. A Decrement and Jump on Not Zero instruction provides for loop control.

2. Centered around the CPU and memory, the 8096’s extensive I/O subsystems include an analog-to-digital converter, a universal asynchronous receiver-transmitter (UART), high-speed input and output circuitry, and a pulse-width modulation output circuit. Such Intelligent I/O allows the CPU to concentrate not on real-time housekeeping but on high-speed arithmetic and control operations.
Most of the instructions execute in about 0.8 μs; the longest, to normalize a zero, takes 8 μs. All data-reference instructions except Pop, Push, and Normalize are available in byte form, and all such instructions except Jump on Bit and Normalize are available in word form. Table 3 lists some typical 8096 instructions and their run times.

A survey of code frequency usage shows that although most multiplications and divisions are unsigned, a signed form is still necessary. When unsigned multiplication and division instructions are preceded by a 0.8-μs SIGND prefix, they are converted into full two's-complement signed multiplication and division. Either type of operation executes in less than 6 μs. Word multiplications result in a double-word product, and byte multiplications produce a word product. With an instruction called Word Divide, a double-word dividend is divided by a word divisor to produce a word quotient and remainders.

Because jumps and calls are PC-relative, code is easy to relocate. Both Jump and Call instructions are available in a short 2-byte form with an 11-bit displacement. Jump on Bit is a 3-byte instruction with an 8-bit displacement. An indirect jump for the “do-case” is also provided.

In addition to the usual sign-extending (EXT) instructions for byte-to-word and word-to-double-word conversions, the set includes instructions LDBSA and LDBZE, which move a byte into a word with sign or zero extension. Most one- and two-operand forms execute in 1 μs. Conditional jumps run in less than 1.8 μs, and in about 0.8 μs when the jump is not taken.

Shifts, whether by a specific number of bit positions or by a computed number, are provided for all three operand lengths (byte, word, and double word). In a floating-point software package, the mantissas must be aligned before they are added or subtracted, and the results normalized afterwards. Both functions require a software shift loop.

The Normalize instruction and the computer form of the Shift Double Word instruction allow fast software implementations of floating-point arithmetic with up to a 32-bit mantissa. Multibit shift instructions are very useful for scaling operations in scaled-integer arithmetic. Scaled-integer operations are usually faster than floating-point arithmetic in control applications.

In addition to an overflow flag, which is set by each arithmetic instruction, there is an overflow-trap flag. It can be checked at the end of a sequence of instructions to determine whether an overflow has occurred anywhere in the sequence.

The instruction set is complemented by a variety of I/O subsystems for handling virtually any com-

### Table 3. Typical instruction times

<table>
<thead>
<tr>
<th>Address Codeword</th>
<th>Operands</th>
<th>Mnemonics</th>
<th>Run Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>0</td>
<td>CLRC,SETC,DI,EL,CLRT,SIGND</td>
<td>Flag manipulations</td>
</tr>
<tr>
<td>0.8</td>
<td>1</td>
<td>INC,DECLR,NOT,NEGS,SEX</td>
<td>One-operand instructions</td>
</tr>
<tr>
<td>0.8</td>
<td>2*</td>
<td>XOR,ADDC,SUB,AND,ADD,SHBC</td>
<td>Two-operand arithmetics</td>
</tr>
<tr>
<td>0.8</td>
<td>2*</td>
<td>OR,CMP</td>
<td>Two-op arithmetics</td>
</tr>
<tr>
<td>0.8</td>
<td>5*</td>
<td>LD,LDBSE,LDBZE,ST</td>
<td>Load and store registers</td>
</tr>
<tr>
<td>0.8 (not taken)</td>
<td>1</td>
<td>JC,JNC,ETC.</td>
<td>Conditional jumps</td>
</tr>
<tr>
<td>1.0</td>
<td>3*</td>
<td>AND,SUB,ADD</td>
<td>Three-op arithmetics</td>
</tr>
<tr>
<td>1.0 (not taken)</td>
<td>2</td>
<td>JBS,JBC</td>
<td>Jump on bit/jump on not bit</td>
</tr>
<tr>
<td>1.8 (taken)</td>
<td>1</td>
<td>JC,JNC,ETC.</td>
<td>Conditional jumps</td>
</tr>
<tr>
<td>1.6</td>
<td>1</td>
<td>SJMP, JMP, JMP</td>
<td>Unconditional jumps</td>
</tr>
<tr>
<td>1.6 (stack register)</td>
<td>0</td>
<td>PUSHF</td>
<td>Push PSW</td>
</tr>
<tr>
<td>1.6 (stack register)</td>
<td>1*</td>
<td>PUSH</td>
<td>Stack push</td>
</tr>
<tr>
<td>1.8 (taken)</td>
<td>2</td>
<td>JBS,JBC,DJNZ</td>
<td>Jump on bit/decrement and jump</td>
</tr>
<tr>
<td>1.6 + 0.2/shift</td>
<td>2</td>
<td>SHL, SHR, SHRA</td>
<td>Shift instructions</td>
</tr>
<tr>
<td>1.8</td>
<td>0</td>
<td>POPF</td>
<td>Pop PSW</td>
</tr>
<tr>
<td>2.2 + 0.2/shift</td>
<td>2</td>
<td>NORML</td>
<td>Normalize</td>
</tr>
<tr>
<td>2.4</td>
<td>1*</td>
<td>POP</td>
<td>Stack pop</td>
</tr>
<tr>
<td>2.4 (stack register)</td>
<td>1</td>
<td>LCALL,SCALL,RET</td>
<td>Subroutines</td>
</tr>
<tr>
<td>2.4 (stack external)</td>
<td>0</td>
<td>PUSHF</td>
<td>Push PSW</td>
</tr>
<tr>
<td>2.4 (stack external)</td>
<td>1*</td>
<td>PUSH</td>
<td>Stack push</td>
</tr>
<tr>
<td>2.6 (stack external)</td>
<td>0</td>
<td>POPF</td>
<td>Pop PSW</td>
</tr>
<tr>
<td>2.8 (stack external)</td>
<td>1*</td>
<td>POP</td>
<td>Stack pop</td>
</tr>
<tr>
<td>3.0 (stack external)</td>
<td>1</td>
<td>SCALL, CALL</td>
<td>Subroutines</td>
</tr>
<tr>
<td>3.2 (stack external)</td>
<td>0</td>
<td>RET</td>
<td>Subroutine</td>
</tr>
<tr>
<td>3.4</td>
<td>2*</td>
<td>MULB</td>
<td>Byte multiplication</td>
</tr>
<tr>
<td>3.6</td>
<td>2*</td>
<td>DIVB</td>
<td>Byte division</td>
</tr>
<tr>
<td>3.6</td>
<td>3*</td>
<td>MULB</td>
<td>Byte multiplication</td>
</tr>
<tr>
<td>5.2</td>
<td>2*</td>
<td>MUL</td>
<td>Word multiplication</td>
</tr>
<tr>
<td>5.2</td>
<td>2*</td>
<td>DIV</td>
<td>Word division</td>
</tr>
<tr>
<td>5.4</td>
<td>3*</td>
<td>MUL</td>
<td>Word multiplication</td>
</tr>
</tbody>
</table>

*One of these operands may have full address modes
puter peripheral or industrial application (Fig. 2). They include an a-d converter, a UART, timer-counters, and a programmable pulse-width-modulated output.

I/O resources include a-d

The controller contains a complete eight-channel, 10-bit a-d converter. Using successive approximation to achieve high speed—33.6 μs at a 15-MHz clock rate—it handles analog input voltages in the range of 0 to 5 V. An external reference is required and must be connected between the reference voltage and analog ground terminals. The converter generates a vectored interrupt when it completes a conversion cycle, allowing the CPU to have rapid access to the a-d input handler when operating in a multitask environment.

Conversion is initiated by writing to an 8-bit a-d command register. The results of a conversion are read from two 8-bit output data registers. One 8-bit register contains the eight most significant bits, and the other holds the two least significant bits, a 3-bit channel indicator, two unused bits, and a status bit. The status bit, which indicates whether the a-d conversion is still in progress, is typically used in a noninterrupt-driven environment.

Just four bits of the a-d command register are used. Three of the bits specify the channel to be converted, and the fourth specifies the method of initiating an a-d conversion cycle. For example, if the fourth bit is a 1, the cycle begins immediately after writing to the command register. If it is a 0, the high-speed output logic subsystem initiates the conversion. The reason for the option is that many data acquisition algorithms require that conversions occur at specific intervals. This requirement is often difficult to manage through software because of interrupt latency and other conditions. Thus, the high-speed output subsystem provides the proper timing for periodic a-d conversions.

The 8096's UART is virtually a carbon copy of the one on the 8051 microcontroller. One of its 8-bit registers receives data, another transmits data, and another indicates the UART status plus bits to configure it for a specific operating mode. By setting the appropriate bits in the third, or control-status, register, a user can select one of four modes:

- Mode 0 (shift register) is a simple, synchronous mode in which the 8096 provides a clock to synchronize incoming or outgoing data. Mode 0 can also be used to expand the I/O.
- Mode 1 is an 8-bit UART mode in which the eighth bit is used for parity when it is enabled.
- Mode 2 is a 9-bit UART mode in which the ninth bit is used for parity when it is enabled.
- Mode 3 is a 9-bit data/address mode in which the UART transmits and receives nine bits of data. This is useful for implementing a simple multiprocessor intercommunications link in which the ninth bit distinguishes address from data.

The remaining six bits of the control-status register are used for six operations: enabling the receiver section of the UART, enabling parity for both transmission and reception (even parity); storing the ninth bit when in the 9-bit transmitting mode; storing the ninth bit when in the 9-bit receiving mode, indicating that the receiver is ready, and indicating that the transmitter is ready. Also on board are a dedicated 15-bit baud-rate generator and a baud-rate clock that can be driven by either the 8096's crystal oscillator or an input at pin T2CLK. This gives maximum flexibility in setting baud rates.

The pulse-width modulated output can produce a pulse train of variable duty cycle, which can be integrated and clamped to provide an accurate digital-to-analog output function. The PWM circuit operates as follows: The 8096 crystal frequency is divided by three and clocks an 8-bit free-running counter. The counter output connects to one side of an 8-bit comparator; the other side of the comparator is tied to a user-addressable register. When the free-running counter value is the same as the one stored in the addressable register, an R-S flip-flop is set. The flip-flop is also reset when the counter rolls over from a count of 255 to 0. This produces a simple yet accurate variable duty-cycle oscillator, which can be programmed for a variable duty cycle from 0 to 255 in increments of X/256.
16-bit microcontroller

The watchdog timer offers a simple way to recover from a software or hardware error. Essentially a 16-bit free-running counter that is clocked by the CPU clock generator circuitry, the timer is reset by writing a 01E0H followed by a 0E10H to byte location 000AH. If a resetting does not occur at least once every 13.107 ms, the timer will overflow, causing the 8096 to be reset—resetting reinitializes the 8096. This feature makes it virtually impossible for the 8096 to become lost in a program for too long. For development purposes, the reset terminal can be connected to VCC to disable the watchdog timer.

More I/O—and faster

Correlating events in real time is one of the most important considerations in computer-based control system design. Another common requirement is generating pulses and pulse trains to drive actuators. Most single-chip microcontrollers support such operations by having one or more timer/event counters under software control. The 8096, on the other hand, offers a complete integrated subsystem to perform these functions. Called the high-speed I/O unit, it is intended to be an integrated subsystem, but it can be viewed as separate units for input and output.

Figure 3 shows the block diagram of the high-speed input unit. Its major components are a 16-bit timer, a programmable change detector and a first-in, first-out (FIFO) memory. Also included are several registers used by the software to control the high-speed input unit.

The read-only timer is cleared by the system reset and incremented once every eight CPU cycles (every 1.6 μs with a 15-MHz crystal). When the timer overflows—rolls over from FFFFH to 00001H—a status bit is set and an interrupt is generated. The change detector monitors four pins on the 8096 and looks for predefined changes. Change definitions are controlled by the high-speed input unit's mode register, which is set by the software. This register contains a 2-bit field for each of the four high-speed inputs. Using the fields, a programmer can select the type of change for each input. Fields are encoded in one of four ways:

- 00 defines positive transitions divided by 8.
- 01 defines positive transitions.
- 10 defines negative transitions.
- 11 defines positive and negative transitions.

Each high-speed input can be disabled through a control register. When this is done, inputs of the high-speed input unit become available as digital input pins or, if required, two of the pins can be connected to the high-speed output unit.

As the block diagram in Fig. 4 shows, the high-speed output unit uses the same timer as the input unit and also has a 16-bit event counter. The read-only event counter is similar to the timer in that it can be read at any time, generates an overflow-interrupt or status indication, and cannot be written into. It differs from the timer, in that its reset and clock sources, instead of being fixed by hardware, can be selected under software control.

Two of the 8096's pins are dedicated to the event counter. A positive-going pulse on ECRST (Event Counter Reset) clears the counter, and either edge of a pulse applied to ECCLK (Event Counter Clock) increments the counter. A programmer has the option of using HSI2 instead of ECRST or HSI1 instead of ECCLK. These options are available by setting the appropriate bits in the I/O control register. The event counter can also be cleared under software control either directly, by setting a bit in the I/O control register, or indirectly, using the high-speed input unit and Eco.
speed output unit itself.

The FIFO register of the high-speed input unit is replaced by a content-addressable memory in the output unit. The memory contains a file of eight 23-bit registers. The 23 bits are divided into a 16-bit time field and a 7-bit command field. Control logic continually scans each location in the memory to determine whether its time field matches either the timer or the event counter as selected by one of the seven bits in the command field. When a match is found, the remaining six bits in the command field are executed.

What the bits do

The encoding of the command field bits and their functions are shown in Fig. 5. Four-bit channel code selects the output unit's operation. For example, the event counter can be reset or an a-d conversion can be initiated. If one of the high-speed output pins is to be changed, bit 5 of the command field will determine its state. For all of the high-speed output unit's operations, bit 4 determines whether an interrupt is generated upon execution of a command.

The high-speed output unit uses two interrupt vectors, one for the software timers and one for all other functions. When a software timer interrupt occurs, the interrupt service routine can interrogate an I/O status register to determine which of the four timers caused the interrupt. The ability of the command field to trigger an a-d conversion allows measurements to be made at precise moments, an absolute necessity in digital signal processing. Also, the ability to reset a count when it reaches a preset limit allows the simple implementation of a modulo-N counter. This is useful, for example, in a crankshaft position-sensor application that generates 214 pulses per revolution.

The eight locations in the content-addressable memory's file are scanned at the rate of one CPU cycle per location. At a 15-MHz clock rate, all eight locations will be scanned within 1.6 \(\mu s\). A high-speed output unit's command is executed as soon as it finds a time match. As each command is executed, it is removed from the content-addressable memory to make room for a new command, which is sent in from the input holding register.

Because of the extensive functions built into the 8096, a standard 40-lead DIP is far too small; the 8096 is housed in a 68-pin JEDEC package. Alternatively, it is supplied in a 48-pin DIP.\(\square\)
The Single Component
MCS®-48 System

12
CHAPTER 12
THE SINGLE COMPONENT MCS®-48 SYSTEM

12.0 INTRODUCTION

Sections 12.1 through 12.4 describe in detail the functional characteristics of the 8748H and 8749H EPROM, 8048AH/8049AH/8050AH ROM, and 8035AHL/8039AHL/8040-AHL CPU only single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. This chapter is limited to those functions useful in single-chip implementations of the MCS®-48. Chapter 14 discusses functions which allow expansion of program memory, data memory, and input output capability.

12.1 ARCHITECTURE

The following sections break the MCS-48 Family into functional blocks and describe each in detail. The following description will use the 8048AH as the representative product for the family. See Figure 14.1.

12.1.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048AH and can be divided into the following blocks:

- Arithmetic Logic Unit (ALU)
- Accumulator
- Carry Flag
- Instruction Decoder

In a typical operation data stored in the accumulator is combined in the ALU with data from another source on the internal bus (such as a register or I/O port) and the result is stored in the accumulator or another register.

The following is more detailed description of the function of each block.

INSTRUCTION DECODER

The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

ARITHMETIC LOGIC UNIT

The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions:

- Add With or Without Carry
- AND, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

If the operation performed by the ALU results in a value represented by more than 8 bits (overflow of most significant bit), a Carry Flag is set in the Program Status Word.

ACCUMULATOR

The accumulator is the single most important data register in the processor, being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.

12.1.2 Program Memory

Resident program memory consists of 1024, 2048, or 4096 words eight bits wide which are addressed by the program counter. In the 8748H and the 8749H this memory is user programmable and erasable EPROM; in the 8048AH/8049AH/8050AH the memory is ROM which is mask programmable at the factory. The 8035AHL/8039AHL/8040-AHL has no internal program memory and is used with external memory devices. Program code is completely interchangeable among the various versions. To access the upper 2K of program memory in the 8050AH, and other MCS-48 devices, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2K boundary.

There are three locations in Program Memory of special importance as shown in Figure 12.2.

LOCATION 0
Activating the Reset line of the processor causes the first instruction to be fetched from location 0.

LOCATION 3
Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

LOCATION 7
A timer/counter interrupt resulting from timer counter overflow (if enabled) causes a jump to subroutine at location 7.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service subroutine is stored in location 3, and the first word of a timer/counter service routines
is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOV and MOV3 allow easy access to data "lookup" tables.

![Program Memory Map](image)

**Figure 12-2. Program Memory Map**

### 12.1.3 Data Memory

Resident data memory is organized as 64, 128, or 256 by 8-bits wide in the 8048AH, 8049AH and 8050AH. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, as shown in Figure 12-3, the first 8 locations (0–7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24–31 are designated as the working registers in place of locations 0–7 and are then directly addressable. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service subroutines allowing the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note that if this second bank is not used, locations 24–31 are still addressable as general purpose RAM. Since the two RAM pointer Registers R0 and R1 are a part of the working register array, bank switching effectively creates two more pointer registers (R0/and R1/) which can be used with R0 and R1 to easily access up to four separate working areas in RAM at one time. RAM locations (8–23) also serve a dual role in that they contain the program counter stack as explained in Section 12.1.6. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting is less than 8, all stack registers are not required and can be used as general purpose RAM locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

![Data Memory Map](image)

**Figure 12-3. Data Memory Map**

12-3
These graphs are for informational purposes only and are not guaranteed minimums or maximums.

Figure 12-4. "Quasi-bidirectional" Port Structure
12.1.4 Input/Output

The 8048AH has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each which serve as either inputs, outputs or bidirectional ports and 3 “test” inputs which can alter program sequences when tested by conditional jump instructions.

PORTS 1 AND 2

Ports 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, and output, or both even though outputs are statically latched. Figure 12-4 shows the circuit configuration in detail. Each line is continuously pulled up to VCC through a resistive device of relatively high impedance.

This pullup is sufficient to provide the source current for a TTL high level yet can be pulled low by a standard TTL gate thus allowing the same pin to be used for both input and output. To provide fast switching times in a “0” to “1” transition a relatively low impedance device is switched in momentarily (= 1/5 of a machine cycle) whenever a “1” is written to the line. When a “0” is written to the line a low impedance device overrides the light pullup and provides TTL current sinking capability. Since the pulldown transistor is a low impedance device a “1” must first be written to any line which is to be used as an input. Reset initializes all lines to the high impedance “1” state.

It is important to note that the ORL and the ANL are read/write operations. When executed, the µC “reads” the port, modifies the data according to the instruction, then “writes” the data back to the port. The “writing” (essentially an OUTL instruction) enables the low impedance pull-up momentarily again even if the data was unchanged from a “1.” This specifically applies to configurations that have inputs and outputs mixed together on the same port. See also section 13.7.

BUS

Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobos. If the bidirectional feature is not needed, Bus can serve as either a statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed however.

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding RD and WR output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the WR output line and output data is valid at the trailing edge of WR. A read of the port generates a pulse on the RD output line and input data must be valid at the trailing edge of RD. When not being written or read, the BUS lines are in a high impedance state. See also sections 13.6 and 13.7.

12.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and INT. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and INT pins have other possible functions as well. See the pin description in Section 12.2.

12.1.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented suing pairs of registers in the Data Memory Array. Only 10, 11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048AH, 8049AH, or 8050AH, while the most significant bits can be used for external Program Memory fetches. See Figure 12.5. The Program Counter is initialized to zero by activating the Reset line.

![Figure 12-5. Program Counter](image-url)

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack as shown in Figure 12-6. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW).
**SINGLE COMPONENT MCS®-48 SYSTEM**

The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine status after a power down sequence.

**Figure 12-6. Program Counter Stack**

Data RAM locations 8-23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in Figure 12-6. The Stack Pointer when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines within subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the Stack Pointer to be decremented and the contents of the resulting register pair to be transferred to the Program Counter.

**12.1.7 Program Status Word**

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). Figure 12-7 shows the information available in the word.

**Figure 12-7. Program Status Word (PSW)**

The upper four bits of PSW are stored in the Program Counter Stack with every call to subroutine or interrupt vector and are optionally restored upon return with the RETR instruction. The RET return instruction does not update PSW.

The PSW bit definitions are as follows:

- **Bits 0-2:** Stack Pointer bits ($S_0$, $S_1$, $S_2$)
- **Bit 3:** Not used ("1" level when read)
- **Bit 4:** Working Register Bank Switch Bit (BS)  
  0 = Bank 0  
  1 = Bank 1
- **Bit 5:** Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JFO.
- **Bit 6:** Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.
- **Bit 7:** Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

**12.1.8 Conditional Branch Logic**

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the conditions that are listed in Table 12-1 can effect a change in the sequence of the program execution.
interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "wire ORing" of several interrupt sources at the input pin. Figure 12-8 shows the interrupt logic of the 8048AH. The Interrupt line is sampled every instruction cycle and when detected causes a "call to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. On 2-cycle instructions the interrupt line is sampled on the 2nd cycle only. INT must be held low for at least 3 machine cycles to ensure proper interrupt operations. As in any CALL to subroutine, the Program Counter and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR reenables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (ones less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

### Interrupt Timing

The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until enabled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the 8048AH may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The INT pin may also be tested using the conditional jump instruction JNI. This instruction may be used to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled, INT may be used as another test input like T0 and T1.

### Time/Counter

The 8048AH contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter. The timer/event counter is shown in Figure 12-9.

### COUNTER

The 8-bit binary counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content may be affected by Reset and should be initialized by software. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once started the counter will increment to this maximum count (FF) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNT1 and DIS TCNT1 instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored.

If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to
1. WHEN INTERRUPT IN PROGRESS FLIP-FLOP IS SET ALL FURTHER INTERRUPTS ARE LOCKED OUT INDEPENDENT OF STATE OF EITHER INTERRUPT ENABLE FLIP-FLOP.
2. WHILE TIMER INTERRUPTS ARE DISABLED TIMER OVERFLOW FF WILL NOT STORE ANY OVERFLOW THAT OCCURS. TIMER FLAG WILL BE SET, HOWEVER.

Figure 12-8. Interrupt Logic
location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and immediately be recognized upon return from the service routine. The pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TCNT1 instruction.

AS AN EVENT COUNTER

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3 or in later MCS-48 devices in state time 4. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held low for at least 1 machine cycle to insure it won’t be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every 5.7 μsec when using an 8 MHz crystal) — there is no minimum frequency. T1 input must remain high for at least 1/5 machine cycle after each transition.

AS A TIMER

Execution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived bypassing the basic machine cycle clock through a +32 prescaler. The prescaler is reset during the START T instruction. The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counts may be achieved by accumulating multiple overflows in a register under software control. For time resolution less than 1 count an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or “fine tuning” of larger delays can be easily accomplished by software delay loops.

Often a serial link is desirable in an MCS-48 family member. Table 12-2 lists the timer counts and cycles needed for a specific baud rate given a crystal frequency.

12.1.11 Clock and Timing Circuits

Timing generation for the 8048AH is completely self-contained with the exception of a frequency reference which can be XTAL, ceramic resonator, or external clock source. The Clock and Timing circuitry can be divided into the following functional blocks.

OSCILLATOR

The on-board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 11 MHz. The X1 external pin is the input to the amplifier stage while X2 is the output. A crystal or ceramic resonator connected between X1 and X2 provides the feedback and phase shift required for oscillation. If an accurate frequency reference is not required, ceramic resonator may be used in place of the crystal.

For accurate clocking, a crystal should be used. An externally generated clock may also be applied to X1–X2 as the frequency source. See the data sheet for more information.
## Table 12-2. Baud Rate Generation

<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Tcy</th>
<th>T0 Prr(1/5 Tcy)</th>
<th>Timer Prescaler (32 Tcy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3.75µs</td>
<td>750ns</td>
<td>120µs</td>
</tr>
<tr>
<td>6</td>
<td>2.50µs</td>
<td>500ns</td>
<td>80µs</td>
</tr>
<tr>
<td>8</td>
<td>1.88µs</td>
<td>375ns</td>
<td>60.2µs</td>
</tr>
<tr>
<td>11</td>
<td>1.36µs</td>
<td>275ns</td>
<td>43.5µs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>75 + 24 Cycles, .01% Error</td>
<td>113 + 20 Cycles, .01% Error</td>
<td>151 + 3 Cycles, .01% Error</td>
</tr>
<tr>
<td>300</td>
<td>27 + 24 Cycles, .1% Error</td>
<td>41 + 21 Cycles, .03% Error</td>
<td>55 + 13 Cycles, .01% Error</td>
</tr>
<tr>
<td>1200</td>
<td>6 + 30 Cycles, .1% Error</td>
<td>10 + 13 Cycles, .1% Error</td>
<td>12 + 27 Cycles, .06% Error</td>
</tr>
<tr>
<td>1800</td>
<td>4 + 20 Cycles, .1% Error</td>
<td>6 + 30 Cycles, .1% Error</td>
<td>9 + 7 Cycles, .17% Error</td>
</tr>
<tr>
<td>2400</td>
<td>3 + 15 Cycles, .1% Error</td>
<td>5 + 6 Cycles, .4% Error</td>
<td>6 + 24 Cycles, .29% Error</td>
</tr>
<tr>
<td>4800</td>
<td>1 + 23 Cycles, 1.0% Error</td>
<td>2 + 19 Cycles, .4% Error</td>
<td>3 + 14 Cycles, .74% Error</td>
</tr>
</tbody>
</table>

### STATE COUNTER

The output of the oscillator is divided by 3 in the State Counter to create a clock which defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENTO CLK instruction. The output of CLK on T0 is disabled by Reset of the processor.

### CYCLE COUNTER

CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states as shown in Figure 12-10. Figure 12-11 shows the different internal operations as divided into the machine states. This clock is called Address Latch Enable (ALE) because of its function in MCS-48 systems with external memory. It is provided continuously on the ALE output pin.

#### 12.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pull-up device which in combination with an external 1 µf capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 12-12. If the reset pulse is generated externally the RESET pin must be held low for at least 10 milliseconds after the power supply is within tolerance. Only 5 machine cycles (6.8 µs @ 11 MHz) are required if power is already on and the oscillator has stabilized. ALE and PSEN (if EA = 1) are active while in Reset.

Reset performs the following functions:

1) Sets program counter to zero.

2) Sets stack pointer to zero.

3) Selects register bank 0.

4) Selects memory bank 0.

5) Sets BUS to high impedance state (except when EA = 5V).

6) Sets Ports 1 and 2 to input mode.

7) Disables interrupts (timer and external).

8) Stops timer.

9) Clears timer flag.

10) Clears F0 and F1.

11) Disables clock output from T0.
12.1.13 Single-Step

This feature, as pictured in Figure 12-13, provides the user with a debug capability in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower half of Port 2. The user can therefore follow the program through each of the instruction steps. A timing diagram, showing the interaction between output ALE and input SS, is shown. The BUS buffer contents are lost during single step; however, a latch may be added to reestablish the lost I/O capability if needed. Data is valid at the leading edge of ALE.
<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN A,P</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>--</td>
<td>READ PORT</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>OUTL_P,A</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>OUTPUT TO PORT</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>ANL P, = DATA</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>READ PORT</td>
<td>FETCH IMMEDIATE DATA</td>
<td>--</td>
<td>INCREMENT PROGRAM COUNTER</td>
</tr>
<tr>
<td>ORL P, = DATA</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>READ PORT</td>
<td>FETCH IMMEDIATE DATA</td>
<td>--</td>
<td>INCREMENT PROGRAM COUNTER</td>
</tr>
<tr>
<td>INS A, BUS</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>--</td>
<td>READ PORT</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>OUTL BUS, A</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>OUTPUT TO PORT</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>ANL BUS, = DATA</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>READ PORT</td>
<td>FETCH IMMEDIATE DATA</td>
<td>--</td>
<td>INCREMENT PROGRAM COUNTER</td>
</tr>
<tr>
<td>ORL BUS, = DATA</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*INCREMENT</td>
<td>TIMER</td>
<td>READ PORT</td>
<td>FETCH IMMEDIATE DATA</td>
<td>--</td>
<td>INCREMENT PROGRAM COUNTER</td>
</tr>
<tr>
<td>MOV @ R,A</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>OUTPUT RAM ADDRESS</td>
<td>INCREMENT TIMER</td>
<td>OUTPUT DATA TO RAM</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVX A, @R</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>OUTPUT RAM ADDRESS</td>
<td>INCREMENT TIMER</td>
<td>--</td>
<td>READ DATA</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>MOVD A,Pi</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>OUTPUT CODE/ADDRESS</td>
<td>INCREMENT TIMER</td>
<td>--</td>
<td>READ P2 LOWER</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>MOVDPi,A</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>OUTPUT CODE/ADDRESS</td>
<td>INCREMENT TIMER</td>
<td>OUTPUT DATA TO P2 LOWER</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANLD P,A</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>OUTPUT CODE/ADDRESS</td>
<td>INCREMENT TIMER</td>
<td>OUTPUT DATA</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORLD P,A</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>OUTPUT CODE/ADDRESS</td>
<td>INCREMENT TIMER</td>
<td>OUTPUT DATA</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J(CONDITIONAL)</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>SAMPLE CONDITION</td>
<td>*INCREMENT SAMPLE</td>
<td>FETCH IMMEDIATE DATA</td>
<td>--</td>
<td>UPDATE PROGRAM COUNTER</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>STRT T</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*</td>
<td>START COUNTER</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRT CNT</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*</td>
<td>--</td>
<td>STOP COUNTER</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>STOP TCNT</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*</td>
<td>ENABLE INTERRUPT</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENI</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*</td>
<td>DISABLE INTERRUPT</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISI</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*</td>
<td>ENABLE CLOK</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTO CLK</td>
<td>FETCH</td>
<td>INCREMENT</td>
<td>PROGRAM COUNTER</td>
<td>--</td>
<td>*</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.

(1) IN LATER MCS-48 DEVICES T1 IS SAMPLED IN S4.
The 8048AH operates in a single-step mode as follows:

1. The processor is requested to stop by applying a low level on SS.
2. The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.
3. The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port :2.
4. SS is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.
5. To stop the processor at the next instruction SS must be brought low again soon after ALE goes low. If SS is left high the processor remains in a “Run” mode.

A diagram for implementing the single-step function of the 8748H is shown in Figure 12-13. D-type flip-flop with preset and clear is used to generate SS. In the run mode SS is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single step, preset is removed allowing ALE to bring SS low via the clear input. ALE should be buffered since the clear input of an SN7474 is the equivalent of 3 TTL loads. The processor is now in the stopped state. The next instruction is initiated by clocking a ‘1’ into the flip-flop. This ‘1’ will not appear on SS unless ALE is high removing clear from the flip-flop. In response to SS going high the processor begins an instruction fetch which brings ALE low resetting SS through the clear input and causing the processor to again enter the stopped state.

12.1.14 Power Down Mode

Extra circuitry has been added to the 8048AH/8049AH/8050AH ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

V_CC serves as the 5V supply pin for the bulk of circuitry while the V_DD pin supplies only the RAM array. In normal operation both pins are a 5V while in standby, V_CC is at ground and V_DD is maintained at its standby value. Applying Reset to the processor through the RESET pin inhibits any access to the RAM by the processor and guarantees that RAM cannot be inadvertently altered as power is removed from V_CC.

A typical power down sequence (Figure 12-14) occurs as follows:

1. Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048AH to save all necessary data before V_CE falls below normal operating limits.
2. Power fail signal is used to interrupt processor and vector it to a power fail service routine.
3. Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the V_DD pin and indicate to external circuitry that power fail routine is complete.
4. Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until V_CC is at ground level.

Recovery from the Power Down mode can occur as any other power-on sequence with an external capacitor on the Reset input providing the necessary delay. See the previous section on Reset.
Figure 12-13. Single Step Operation
12.1.15 External Access Mode

Normally the first 1K (8048AH), 2K (8049AH), or 4K (8050AH) words of program memory are automatically fetched from internal ROM or EPROM. The EA input pin however allows the user to effectively disable internal program memory by forcing all program memory fetches to reference external memory. The following chapter explains how access to external program memory is accomplished.

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice — a diagnostic routine for instance. In addition, section 12.4 explains how internal program memory can be read externally, independent of the processor. A “1” level on EA initiates the external access mode. For proper operation, Reset should be applied while the EA input is changed.

12.1.16 Sync Mode

The 8048AH, 8049AH, 8050AH has incorporated a new SYNC mode. The Sync mode is provided to ease the design of multiple controller circuits by allowing the designer to force the device into known phase and state time. The SYNC mode may also be utilized by automatic test equipment (ATE) for quick, easy, and efficient synchronizing between the tester and the DUT (device under test).

SYNC mode is enabled when SS’ pin is raised to high voltage level of +12 volts. To begin synchronization, T0 is raised to 5 volts at least four clocks cycles after SS’. T0 must be high for at least four X1 clock cycles to fully reset the prescaler and time state generators. T0 may then be brought down with the rising edge of X1. Two clock cycles later, with the rising edge of X1, the device enters into Time State 1, Phase 1, SS’ is then brought down to 5 volts 4 clocks later after T0. RESET’ is allowed to go high 5 iCY (75 clocks) later for normal execution of code. See Figure 12-15.

12.1.17 Idle Mode

Along with the standard power down, the 80C438, 80C49, 80C50 has added an IDLE mode instruction (01H) to give even further flexibility and power management. In the IDLE mode, the CPU is frozen while the oscillator, RAM, timer, and the interrupt circuitry remains fully active.

When the IDL instruction (01H) is decoded, the clock to the CPU is stopped. CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all the registers maintain their data throughout idle.

Externally, the following occurs during idle:

1) The ports remain in the logical state they were in when idle was executed.
2) The bus remains in the logical state it was in when idle was executed if the bus was latched.
   If the bus was in a high Z condition or if external program memory is used the bus will remain in the float state.
3) ALE remains in the inactive state (low).
4) RD’, WR’, PROG’, and PSEN’ remains in the inactive state (high).
5) T0 outputs clock if enabled.

There are three ways of exiting idle. Activating any enabled interrupt (external or timer) will cause the CPU to vector to the appropriate interrupt routine. Following a RETR instruction, program execution will resume at the instruction following the address that contained the IDL instruction.

The F0 and F1 flags may be used to give an indication if the interrupt occurred during normal program execution or during idle. This is done by setting or clearing the flags before going into idle. The interrupt service routine can examine the flags and act accordingly when idle is terminated by an interrupt.

Resetting the device can also terminate idle. Since the oscillator is already running, five machine cycles are all that is required to insure proper machine operation.
### 12.2 PIN DESCRIPTION

The MCS-48 processors are packaged in 40 pin Dual In-Line Packages (DIP's). Table 12-3 is a summary of the functions of each pin. Figure 12-16 is the logic symbol for the 8048AH product family. Where it exists, the second paragraph describes each pin's function in an expanded MCS-48 system. Unless otherwise specified, each input is TTL compatible and each output will drive one standard TTL load.

---

#### Table 12-3: Pin Functions of the 8048AH

<table>
<thead>
<tr>
<th>Pin Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL</td>
<td>Oscillator Crystal Input</td>
</tr>
<tr>
<td>RESET</td>
<td>Reset Input</td>
</tr>
<tr>
<td>SINGLE STEP</td>
<td>Single Step Input</td>
</tr>
<tr>
<td>EXTERNAL MEM</td>
<td>External Memory Input</td>
</tr>
<tr>
<td>TEST</td>
<td>Test Input</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>Interrupt Input</td>
</tr>
<tr>
<td>BUS</td>
<td>Bus Inputs and Outputs</td>
</tr>
<tr>
<td>PORT #1</td>
<td>Data Input</td>
</tr>
<tr>
<td>PORT #2</td>
<td>Data Output</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>+5V</td>
<td>Power Supply</td>
</tr>
</tbody>
</table>

---

#### Figure 12-15. Sync Mode Timing

<table>
<thead>
<tr>
<th>TIME STATE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>12V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SS .5V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T0 .5V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALE 0V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESET 0V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

#### Figure 12-16. 8048AH and 8049AH Logic Symbol
Table 12-3. Pin Description

<table>
<thead>
<tr>
<th>Designation</th>
<th>Pin Number*</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>V&lt;sub&gt;SS&lt;/sub&gt;</td>
<td>20</td>
<td>Circuit GND potential</td>
</tr>
<tr>
<td>V&lt;sub&gt;DD&lt;/sub&gt;</td>
<td>26</td>
<td>Programming power supply; 21V during program for the 8748H/8749H; +5V during operation for both ROM and EPROM. Low power standby pin in 8048AH and 8049AH/8050AH ROM versions.</td>
</tr>
<tr>
<td>V&lt;sub&gt;CC&lt;/sub&gt;</td>
<td>40</td>
<td>Main power supply; +5V during operation and during 8748H and 8749H programming.</td>
</tr>
<tr>
<td>PROG</td>
<td>25</td>
<td>Program pulse; +18V input pin during 8748H/8749H programming. Output strobe for 8243 I/O expander.</td>
</tr>
<tr>
<td>P10–P17 (Port 1)</td>
<td>27–34</td>
<td>8-bit quasi-bidirectional port. (Internal Pullup ≈ 50KΩ)</td>
</tr>
<tr>
<td>P20–P27 (Port 2)</td>
<td>21–24</td>
<td>8-bit quasi-bidirectional port. (Internal Pullup ≈ 50KΩ)</td>
</tr>
<tr>
<td>P20–P23</td>
<td>35–38</td>
<td>P20–P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.</td>
</tr>
<tr>
<td>D0–D7 (BUS)</td>
<td>12–19</td>
<td>True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.</td>
</tr>
<tr>
<td>I0</td>
<td>1</td>
<td>Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENTO CLK instruction. T0 is also used during programming and sync mode.</td>
</tr>
<tr>
<td>I1</td>
<td>39</td>
<td>Input pin testable using the JT1, and JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction. (See Section 2.1.10)</td>
</tr>
<tr>
<td>INT</td>
<td>6</td>
<td>Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. (Active low) Interrupt must remain low for at least 3 machine cycles to ensure proper operation.</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. (Active low) Used as a Read Strobe to External Data Memory.</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>Input which is used to initialize the processor. Also used during EPROM programming and verification. (Active low) (Internal pullup ≈ 80K Ω)</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>Output strobe during a BUS write. (Active low) Used as write strobe to external data memory.</td>
</tr>
<tr>
<td>ALE</td>
<td>11</td>
<td>Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.</td>
</tr>
</tbody>
</table>
Table 12-3. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Designation</th>
<th>Pin Number*</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSEN</td>
<td>9</td>
<td>Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)</td>
</tr>
<tr>
<td>SS</td>
<td>5</td>
<td>Single step input can be used in conjunction with ALE to “single step” the processor through each instruction. (Active low) (Internal pullup ≈ 300KΩ) +12V for sync modes (See 2.1.16)</td>
</tr>
<tr>
<td>EA</td>
<td>7</td>
<td>External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) +12V for 8048AH/8049AH/8050AH program verification and +18V for 8748H/8749H program verification (Internal pullup = 10MΩ on 8048AH/8049AH/8035AHL/8039AHL/8050AH/8040AHL)</td>
</tr>
<tr>
<td>XTAL1</td>
<td>2</td>
<td>One side of crystal input for internal oscillator. Also input for external source.</td>
</tr>
<tr>
<td>XTAL2</td>
<td>3</td>
<td>Other side of crystal/external source input.</td>
</tr>
</tbody>
</table>

*Unless otherwise stated, inputs do not have internal pullup resistors. 8048AH, 8049AH, 8050AH, 8040AHL

12.3 PROGRAMMING, VERIFYING AND ERASING EPROM

The internal Program Memory of the 8748H and the 8749H may be erased and reprogrammed by the user as explained in the following sections. See also the 8748H and 8749H data sheets.

12.3.1 Programming/Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. This programming algorithm applies to both the 8748H and 8749H. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL 1</td>
<td>Clock Input (3 to 4 MHz)</td>
</tr>
<tr>
<td>Reset</td>
<td>Initialization and Address Latching</td>
</tr>
<tr>
<td>Test 0</td>
<td>Selection of Program (0V) or Verify (5V) Mode</td>
</tr>
<tr>
<td>EA</td>
<td>Activation of Program/Verify Modes</td>
</tr>
<tr>
<td>BUS</td>
<td>Address and Data Input Data Output During Verify</td>
</tr>
<tr>
<td>P20-1</td>
<td>Address Input for 8748H</td>
</tr>
<tr>
<td>P20-2</td>
<td>Address Input for 8749H</td>
</tr>
<tr>
<td>VDD</td>
<td>Programming Power Supply</td>
</tr>
<tr>
<td>PROG</td>
<td>Program Pulse Input</td>
</tr>
<tr>
<td>P10–P11</td>
<td>Tied to ground (8749H only)</td>
</tr>
</tbody>
</table>

8748H AND 8749H ERASURE CHARACTERISTICS

The erasure characteristics of the 8748H and 8749H are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8748H and 8749H in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8748H or 8749H is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the 8748H window to prevent unintentional erasure.

When erased, bits of the 8748H and 8749H Program Memory are in the logic “0” state.

The recommended erasure procedure for the 8748H and 8749H is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm². The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm² power rating. The 8748H and 8749H should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter in their tubes and this filter should be removed before erasure.
SINGLE COMPONENT MCS-48 SYSTEM

COMBINATION PROGRAM/VERIFY MODE (EPROM's ONLY)

18V
EA
5V
TO
RESET

DB0-DB7
ADDRESS (0-7) VALID
DATA TO BE
PROGRAMMED VALID
DATA
VALID
NEXT ADDR
VALID

P20-P22
LAST ADDRESS
ADDRESS (8-10) VALID
NEXT ADDRESS

VERIFICATION MODE (ROM/EPROM)

EA

NOTE:
1. PROG MUST FLOAT IF EA IS LOW (I.E., \( \neq 18V \)).

*T0 ON EPROM ONLY.

Figure 12-17. Program/Verify Sequence for 8749H/8748H
CHAPTER 13
EXPANDED MCS®-48 SYSTEM

13.0 INTRODUCTION

If the capabilities resident on the single-chip 8048AH/8748H/8035AH/8049AH/8749H/8039AH are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

- Program Memory to 4K words
- Data Memory to 320 words (384 words with 8049AH)
- I/O by unlimited amount
- Special Functions using 8080/8085AH peripherals

By using bank switching techniques, maximum capability is essentially unlimited. Bank switching is discussed later in the chapter. Expansion is accomplished in two ways:

1) Expander I/O — A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.

2) Standard 8085 Bus — One port of the 8048AH/8049AH is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS®-80/85 microcomputer family.

MCS-48 systems can be configured using either or both of these expansion features to optimize system capabilities to the application.

Both expander devices and standard memories and peripherals can be added in virtually any number and combination required.

13.1 EXPANSION OF PROGRAM MEMORY

Program Memory is expanded beyond the resident 1K or 2K words by using the 8085 BUS feature of the MCS®-48. All program memory fetches from the addresses less than 1024 on the 8048AH and less than 2048 on the 8049AH occur internally with no external signals being generated (except ALE which is always present). At address 1024 on the 8048AH, the processor automatically initiates external program memory fetches.

13.1.1 Instruction Fetch Cycle (External)

As shown in Figure 13-1, for all instruction fetches from addresses of 1024 (2048) or greater, the following will occur:

1) The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.
2) Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.
3) Program Store Enable (PSEN) indicates that an external instruction fetch is in progress and serves to enable the external memory device.
4) BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

![Figure 13-1. Instruction Fetch from External Program Memory](image)

All instruction fetches, including internal addresses, can be forced to be external by activating the EA pin of the 8048AH/8049AH/8050AH. The 8035AH/8039AH/8040AH processors without program memory always operate in the external program memory mode (EA = 5V).

13.1.2 Extended Program Memory Addressing (Beyond 2K)

For programs of 2K words or less, the 8048AH/8049AH addresses program memory in the conventional manner. Addresses beyond 2047 can be reached by executing a program memory bank switch instruction (SEL MBO, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2K range and at the same time prevents the user from inadvertently crossing the 2K boundary.

PROGRAM MEMORY BANK SWITCH

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11); see Figure 13-2. Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1
instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch which occurs during the next branch instruction encountered. Since all twelve bits of the program counter, including bit 11, are stored in the stack, when a Call is executed, the user may jump to subroutines across the 2K boundary and the proper bank will be restored upon return. However, the bank switch flip-flop will not be altered on return.

13.1.3 Restoring I/O Port Information

Although the lower half of Port 2 is used to output the four most significant bits of address during an external program memory fetch, the I/O information is still output during certain portions of each machine cycle. I/O information is always present on Port 2’s lower 4 bits at the rising edge of ALE and can be sampled or latched at this time.

13.1.4 Expansion Examples

Shown in Figure 13-3 is the addition of 2K words of program memory using an 2716A 2K x 8 ROM to give a total of 3K words of program memory. In this case no chip select decoding is required and PSEN enables the memory directly through the chip select input. If the system requires only 2K of program memory, the same configuration can be used with an 8035AHL substituted for the 8048AH. The 8049AH would provide 4K of program memory with the same configuration.
Figure 13-4 shows how the 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048AH without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K x 8 program memory, the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; therefore the RD and WR outputs of the 8048AH are required. See the following section on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.

### 13.2 EXPANSION OF DATA MEMORY

Data Memory is expanded beyond the resident 64 words by using the 8085AH type bus feature of the MCS®-48.

#### 13.2.1 Read/Write Cycle

All address and data is transferred over the 8 lines of BUS. As shown in Figure 13-5, a read or write cycle occurs as follows:

---

Figure 13-4. External Program Memory Interface

Figure 13-5. External Data Memory Timings
1) The contents of register R0 or R1 is outputed on BUS.
2) Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.
3) A read (RD) or write (WR) pulse on the corresponding output pins of the 8048AH indicates the type of data memory access in progress. Output data is valid at the trailing edge of WR and input data must be valid at the trailing edge of RD.
4) Dat (8 bits) is transferred in or out over BUS.

13.2.2 Addressing External Data Memory
External Data Memory is accessed with its own two-cycle move instructions. MOVXA, @R and MOVX@R, A, which transfer 8 bits of data between the accumulator and the external memory location addressed by the contents of one of the RAM Pointer Registers R0 and R1. This allows 256 locations to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the 8048AH.

13.2.3 Examples of Data Memory Expansion
Figure 13-6 shows how the 8048AH can be expanded using the 8155 memory and I/O expanding device. Since the 8155 has an internal 8-bit address latch, it can interface directly to the 8048AH without the use of an external latch. The 8155 provides an additional 256 words of static data memory and also includes 22 I/O lines and a 14-bit timer. See the following section on I/O expansion and the 8155 data sheet for more details on these additional features.

13.3 EXPANSION OF INPUT/OUTPUT
There are four possible modes of I/O expansion with the 8048AH: one using a special low-cost expander, the 8243; another using standard MCS-80/85 I/O devices; and a third using the combination memory I/O expander devices the 8155, 8355, and 8755. It is also possible to expand using standard TTL devices as shown in Chapter 5.

13.3.1 I/O Expander Device
The most efficient means of I/O expansion for small systems is the 8243 I/O Expander Device which requires only 4 port lines (lower half of Port 2) for communication with the 8048AH. The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports #4–7 (see Figure 13-7). The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port

A 4-bit transfer from a port to the lower half of the Accumulator sets the most significant four bits to zero. All communication between the 8048AH and the 8243 occurs over Port 2 lower (P20–P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles: The first containing the "op code" and port address, and the second containing the actual 4 bits of data.

Figure 13-6. 8048AH Interface to 256 x 8 Standard Memories
EXPANDED MCS®-48 SYSTEM

13.3.2 I/O Expansion with Standard Peripherals

Standard MCS-80/85 type I/O devices may be added to the MCS®-48 using the same bus and timing used for Data Memory expansion. Figure 13-8 shows an example of how an 8048AH can be connected to an MCS-85 peripheral. I/O devices reside on the Data Memory bus and in the data memory address space and are accessed with the same MOVX instructions. (See the previous section on data memory expansion for a description of timing.) The following are a few of the Standard MCS-80 devices which are very useful in MCS®-48 systems:

- 8214 Priority Interrupt Encoder
- 8251 Serial Communications Interface
- 8255 General Purpose Programmable I/O
- 8279 Keyboard/Display Interface
- 8253 Interval Timer

13.3.3 Combination Memory and I/O Expanders

As mentioned in the sections on program and data memory expansion, the 8355/8755 and 8155 expanders also contain I/O capability.

A high to low transition of the PROG line indicates that address is present, while allow to high transition indicates the presence of data. Additional 8243's may be added to the four-bit bus and chip selected using additional output lines from the 8048AH/8748H.

I/O PORT CHARACTERISTICS

Each of the four 4-bit ports of the 8243 can serve as either input or output and can provide high drive capability in both the high and low state.
8355/8755: These two parts of ROM and EPROM equivalents and therefore contain the same I/O structure. I/O consists of two 8-bit ports which normally reside in the external data memory address space and are accessed with MOVX instructions. Associated with each port is an 8-bit Data Direction Register which defines each bit in the port as either an input or an output. The data direction registers are directly addressable, thereby allowing the user to define under software control each individual bit of the ports as either input or output. All outputs are statically latched and double buffered. Inputs are not latched.

8155/8156: I/O on the 8155/8156 is configured as two 8-bit programmable I/O ports and one 6-bit programmable port. These three registers and a Control/Status register are accessible as external data memory with the MOVX instructions. The contents of the control register determines the mode of the three ports. The ports can be programmed as input or output with or without associated handshake communication lines. In the handshake mode, lines of the six-bit port become input and output strobes for the two 8-bit ports. Also included in the 8155 is a 14-bit programmable timer. The clock input to the timer and the timer overflow output are available on external pins. The timer can be programmed to stop on terminal count or to continuously reload itself. A square wave or pulse output on terminal count can also be specified.
I/O EXPANSION EXAMPLES

Figure 13-9 shows the expansion of I/O using multiple 8243’s. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048AH output lines. Two output lines and a decoder could also be used to address the four chips. Large numbers of 8243’s would require a chip select decoder chip such as the 8205 to save 110 pins.

Figure 13-10 shows the 8048AH interface to a standard MCS®-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40-pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS®-80 peripherals with an 8-bit bidirectional data bus, a RD and WR input for Read/Write control, a CS (chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.

![Figure 13-10. Interface to MCS®-80 Peripherals](image)

Interconnection to the 8048AH is very straightforward with BUS, RD, and WR connecting directly to the corresponding pins on the 8255. The only design consideration is the way in which the internal registers of the 8255 are to be addressed. If the registers are to be addressed as external data memory using the MOVX instructions, the appropriate number of address bits (in this case, 2) must be latched on BUS using ALE as described in the section on external data memories. If only a single device is connected to BUS, the 8255 may be continuously selected by grounding CS. If multiple 8255’s are used, additional address bits can be latched and used as chip selects.

A second addressing method eliminates external latches and chip select decoders by using output port lines as address and chip select lines directly. This method, of course, requires the setting of an output port with address information prior to executing a MOVX instruction.

13.4 MULTI-CHIP MCS®-48 SYSTEMS

Figure 13-11 shows the addition of two memory expanders to the 8048AH, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the addressing of the various memories and I/O ports. Note that in this configuration address lines A_{10} and A_{11} have been ORed to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time, there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and A_{11} connected directly to the CE (instead of CE) input of the 8355; however, this would create a 1K word “hole” in the program memory by causing the 8355 to be active in the 2K and 4K range instead of the normal 1K to 3K range.

In this system the various locations are addressed as follows:

- Data RAM — Addresses 0 to 255 when Port 2 Bit 0 has been previously set = 1 and Bit 1 set = 0
- RAM I/O — Addresses 0 to 3 when Port 2 Bit 0 = 1 and Bit 1 = 1
- ROM I/O — Addresses 0 to 3 when Port 2 Bit 2 or Bit 3 = 1

See the memory map in Figure 13-12.
13.5 MEMORY BANK SWITCHING

Certain systems may require more than the 4K words of program memory which are directly addressable by the program counter or more than the 256 data memory and I/O locations directly addressable by the pointer registers R0 and R1. These systems can be achieved using "bank switching" techniques. Bank switching is merely the selection of various blocks of "banks" of memory using dedicated output port lines from the processor. In the case of the 8048AH, program memory is selected in blocks of 4K words at a time, while data memory and I/O are enabled 256 words at a time.

The most important consideration in implementing two or more banks is the software required to cross the bank boundaries. Each crossing of the boundary requires that the processor first write a control bit to an output port before accessing memory or I/O in the new bank. If program memory is being switched, programs should be organized to keep boundary crossings to a minimum.

Jumping to subroutines across the boundary should be avoided when possible since the programmer must keep track of which bank to return to after completion of the subroutine. If these subroutines are to be nested and accessed from either bank, a software "stack" should be implemented to save the bank switch bit just as if it were another bit of the program counter.

From a hardware standpoint bank switching is very straightforward and involves only the connection of an I/O line or lines as bank enable signals. These enables are ANDed with normal memory and I/O chip select signals to activate the proper bank.

13.6 CONTROL SIGNAL SUMMARY

Table 13 summarizes the instructions which activate the various control outputs of the MCS®-48 processors. During all other instructions these outputs are driven to the active state.
## 13.7 PORT CHARACTERISTICS

### 13.7 BUS Port Operations

The BUS port can operate in three different modes: as a latched I/O port, as a bidirectional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating).

<table>
<thead>
<tr>
<th>Control Signal</th>
<th>When Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD</td>
<td>During MOVX, A, @R or INS Bus</td>
</tr>
<tr>
<td>WR</td>
<td>During MOVX @R, A or OUTL Bus</td>
</tr>
<tr>
<td>ALE</td>
<td>Every Machine Cycle</td>
</tr>
<tr>
<td>PSEN</td>
<td>During Fetch of external program memory (instruction or immediate data)</td>
</tr>
<tr>
<td>PROG</td>
<td>During MOVD, A,P ANLD P,A MOVD P,A ORLD P,A</td>
</tr>
</tbody>
</table>

The latched mode (INS, OUTL) is intended for use in the single-chip configuration where BUS is not begin used as an expander port. OUTL and MOVX instructions can be mixed if necessary. However, a previously latched output will be destroyed by executing a MOVX instruction and BUS will be left in the high impedance state. INS does not put the BUS in a high impedance state. Therefore, the use of MOVX after OUTL to put the BUS in a high impedance state is necessary before an INS instruction intended to read an external word (as opposed to the previously latched value).

OUTL should never be used in a system with external program memory, since latching BUS can cause the next instruction, if external, to be fetched improperly.

### 13.7.2 Port 2 Operations

The lower half of Port 2 can be used in three different ways: as a quasi-bidirectional static port, as an 8243 expander port, and to address external program memory.
In all cases outputs are driven low by an active device and driven high momentarily by a low impedance device and held high by a high impedance device to VCC.

The port may contain latched I/O data prior to its use in another mode without affecting operation of either. If lower Port 2 (P20-3) is used to output address for an external program memory fetch, the I/O information previously latched will be automatically removed temporarily while address is present, then restored when the fetch is complete. However, if lower Port 2 is used to communicate with an 8243, previously latched I/O information will be removed and not restored. After an input from the 8243, P20-3 will be left in the input mode (floating). After an output to the 8243, P20-3 will contain the value written, ANDed, or ORed to the 8243 port.

Figure 13-13. MCS®-48 Expansion Capability
CHAPTER 14
MCS®-48 INSTRUCTION SET

14.0 INTRODUCTION

The MCS®-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 80% are only one byte long. Also, all instructions execute in either one or two cycles and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions.

The MCS-48 microcomputers have been designed to handle arithmetic operations efficiently in both binary and BCD as well as handle the single-bit operations required in control applications. Special instructions have also been included to simplify loop counters, table look-up routines, and N-way branch routines.

14.0.1 Data Transfers

As can be seen in Figure 14.1, the 8-bit accumulator is the central point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e., the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed indirectly via an address stored in either R0 or R1 of the active register bank. R0 and R1 are also used to indirectly address external data memory when it is present. Transfers to and from internal RAM require one cycle, while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-
board timer counter or the accumulator and the Program Status word (PSW). Writing to the PSW alters machine status accordingly and provides a means of restoring status after an interrupt or of altering the stack pointer if necessary.

### 14.0.2 Accumulator Operations

Immediate data, data memory, or the working registers can be added with or without carry to the accumulator. These sources can also be ANDed, ORed, or Exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

In addition, the lower 4 bits of the accumulator can be exchanged with the lower 4-bits of any of the internal RAM locations. This instruction, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides for easy handling of 4-bit quantities, including BCD numbers. To facilitate BCD arithmetic, a Decimal Adjust instruction is included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the required BCD result.

Finally, the accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

Although there is no subtract instruction in the 8048AH, this operation can be easily implemented with three single-byte single-cycle instructions.

A value may be subtracted from the accumulator with the result in the accumulator by:

- Complementing the accumulator
- Adding the value to the accumulator
- Complementing the accumulator

### 14.0.3 Register Operations

The working registers can be accessed via the accumulator as explained above, or can be loaded immediate with constants from program memory. In addition, they can be incremented or decremented or used as loop counters using the decrement and jump, if not zero instruction, as explained under branch instructions.

All Data Memory including working registers can be accessed with indirect instructions via R0 and R1 and can be incremented.

### 14.0.4 Flags

There are four user-accessible flags in the 8048AH: Carry, Auxiliary Carry, F0 and F1. Carry indicates overflow of the accumulator, and Auxiliary Carry is used to indicate overflow between BCD digits and is used during decimal-adjust operation. Both Carry and Auxiliary Carry are accessible as part of the program status word and are stored on the stack during subroutines. F0 and F1 are undedicated general-purpose flags to be used as the programmer desires. Both flags can be cleared or complemented and tested by conditional jump instructions. F0 is also accessible via the Program Status word and is stored on the stack with the carry flags.

### 14.0.5 Branch Instructions

The unconditional jump instruction is two bytes and allows jumps anywhere in the first 2K words of program memory. Jumps to the second 2K of memory (4K words are directly addressable) are made first by executing a select memory bank instruction, then executing the jump instruction. The 2K boundary can only be crossed via a jump or subroutine call instruction, i.e., the bank switch does not occur until a jump is executed. Once a memory bank has been selected all subsequent jumps will be to the selected bank until another select memory bank instruction is executed. A subroutine in the opposite bank can be accessed by a select memory bank instruction followed by a call instruction. Upon completion of the subroutine, execution will automatically return to the original bank; however, unless the original bank is reselected, the next jump instruction encountered will again transfer execution to the opposite bank.

Conditional jumps can test the following inputs and machine status:

- T0 Input Pin
- T1 Input Pin
- INT Input Pin
- Accumulator Zero
- Any bit of Accumulator
- Carry Flag
- F0 Flag
- F1 Flag

Conditional jumps allow a branch to any address within the current page (256 words) of execution. The conditions tested are the instantaneous values at the time the conditional jump is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate zero flag.
The decrement register and jump if not zero instruction combines a decrement and a branch instruction to create an instruction very useful in implementing a loop counter. This instruction can designate any one of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A single-byte indirect jump instruction allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator points to a location in program memory which contains the jump address. The 8-bit jump address refers to the current page of execution. This instruction could be used, for instance, to vector to any one of several routines based on an ASCII character which has been loaded in the accumulator. In this way ASCII key inputs can be used to initiate various routines.

14.0.6 Subroutines

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

14.0.7 Timer Instructions

The 8-bit on board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or an event counter or timer with an external clock applied to the T1 input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

14.0.8 Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 13.1.2.

The working register bank switch instructions allow the programmer to immediately substitute a second 8-register working register bank for the one in use. This effectively provides 16 working registers or it can be used as a means of quickly saving the contents of the registers in response to an interrupt. The user has the option to switch or not to switch banks on interrupt. However, if the banks are switched, the original bank will be automatically restored upon execution of a return and restore status instruction at the end of the interrupt service routine.

A special instruction enables an internal clock, which is the XTAL frequency divided by three to be output on pin TO. This clock can be used as a general-purpose clock in the user’s system. This instruction should be used only to initialize the system since the clock output can be disabled only by application of system reset.

14.0.9. Input/Output Instructions

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows “masks” stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a “1” out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions are executed, a corresponding READ or WRITE pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state. Note that the OUTL, ANL, and the ORL instructions for the BUS are for use with internal program memory only.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports.
I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e., they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

14.1 INSTRUCTION SET DESCRIPTION

The following pages describe the MCS®-48 instruction set in detail. The instruction set is first summarized with instructions grouped functionally. This summary page is followed by a detailed description listed alphabetically by mnemonic opcode.

The alphabetical listing includes the following information:
- Mnemonic
- Machine Code
- Verbal Description
- Symbolic Description
- Assembly Language Example

The machine code is represented with the most significant bit (7) to the left and two byte instructions are represented with the first byte on the left. The assembly language examples are formulated as follows:

Arbitrary
Label: Mnemonic, Operand;
Descriptive Comment
### 8048AH/8748H/8049AH/8050AH/8749H

**Instruction Set Summary**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accumulator</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD A, R</td>
<td>Add register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, @R</td>
<td>Add data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, # data</td>
<td>Add data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A, R</td>
<td>Add register with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, @R</td>
<td>Add data memory with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, # data with carry</td>
<td>Add data memory with carry</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL A, R</td>
<td>And register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, @R</td>
<td>And data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, # data</td>
<td>And immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL A, R</td>
<td>Or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, @R</td>
<td>Or data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, # data</td>
<td>Or immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XRL A, R</td>
<td>Exclusive Or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, @R</td>
<td>Exclusive or data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, # data</td>
<td>Exclusive or immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Input/Output</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN A, P</td>
<td>Input port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUTL A, P</td>
<td>Output A to port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL P, # data</td>
<td>And immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL P, # data</td>
<td>Or immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>*INS A, BUS</td>
<td>Input Bus to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>*OUTL BUS, A</td>
<td>Output Bus to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>*ANL BUS, # data</td>
<td>And immediate to BUS</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>*ORL BUS, # data</td>
<td>Or immediate to BUS</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOVD A, P</td>
<td>Input Expander port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVD P, A</td>
<td>Output A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANLD P, A</td>
<td>And A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORLD P, A</td>
<td>Or A to Expander port</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Registers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INC R</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC @R</td>
<td>Increment data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC R</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Branch</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP addr</td>
<td>Jump unconditional</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JMP @A</td>
<td>Jump indirect</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ R, addr</td>
<td>Decrement register and jump</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JC addr</td>
<td>Jump on carry = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC addr</td>
<td>Jump on carry = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JZ addr</td>
<td>Jump on A Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNZ addr</td>
<td>Jump on A not Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTO addr</td>
<td>Jump on T0 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNTO addr</td>
<td>Jump on T0 = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JT1 addr</td>
<td>Jump on T1 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT1 addr</td>
<td>Jump on T1 = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF0 addr</td>
<td>Jump on F0 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF1 addr</td>
<td>Jump on F1 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTF addr</td>
<td>Jump on timer flag = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNI addr</td>
<td>Jump on INT = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JBb addr</td>
<td>Jump on Accumulator Bit</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Subroutine</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL addr</td>
<td>Jump to subroutine</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RETR</td>
<td>Return and restore status</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Flags</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR C</td>
<td>Clear Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F0</td>
<td>Clear Flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td>Complement Flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F1</td>
<td>Clear Flag 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F1</td>
<td>Complement Flag 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Data Moves</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV A, R</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, @R</td>
<td>Move data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, # data</td>
<td>Move immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV R, A</td>
<td>Move A to register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV @R, A</td>
<td>Move A to data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV R, # data</td>
<td>Move immediate to data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV @R, # data</td>
<td>Move immediate to data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A, PSW</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV PSW, A</td>
<td>Move A to PSW</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Mnemonics copyright Intel Corporation 1983.

*For use with internal memory only.
### MCS®-48 INSTRUCTION SET

#### 8048AH/8748H/8049AH/8050AH/8749H

**Instruction Set Summary (Con't)**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Moves</strong> (Con'td)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XCH A, R</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, @R</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHD A, @R</td>
<td>Exchange nibble of A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVX A, @R</td>
<td>Move external data memory to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVX @R, A</td>
<td>Move A to external data memory</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVP A, @A</td>
<td>Move to A from current page</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVP3 A, @A</td>
<td>Move to A from Page 3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Timer/Counter</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV A, T</td>
<td>Read Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV T, A</td>
<td>Load Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>START</td>
<td>Start Timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>START CNT</td>
<td>Start Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STOP TCNT</td>
<td>Stop Timer/Counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN TCNTI</td>
<td>Enable Timer/Counter Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS TCNTI</td>
<td>Disable Timer/Counter Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Mnemonics copyright Intel Corporation 1983.**
MCS®-48 INSTRUCTION SET
Symbols and Abbreviations Used

A Accumulator
AC Auxiliary Carry
addr 12-Bit Program Memory Address
Bb Bit Designator (b = 0–7)
BS Bank Switch
BUS BUS Port
C Carry
CLK Clock
CNT Event Counter
CRR Conversion Result Register
D Mnemonic for 4-Bit Digit (Nibble)
data 8-Bit Number or Expression
DBF Memory Bank Flip-Flop
F0, F1 Flag 0, Flag 1
I Interrupt
P Mnemonic for “in-page” Operation
PC Program Counter
Pp Port Designator (p = 1, 2 or 4–7)
PSW Program Status Word
Ri Data memory Pointer (i = 0, or 1)
Rr Register Designator (r = 0–7)
SP Stack Pointer
T Timer
TF Timer Flag
T0, T1 Test 0, Test 1
X Mnemonic for External RAM
# Immediate Data Prefix
@ Indirect Address Prefix
$ Current Value of Program Counter
(X) Contents of X
((X)) Contents of Location Addressed by X
← Is Replaced by

Mnemonics copyright Intel Corporation 1983.
MCS®-48 INSTRUCTION SET

ADD A,Rr  Add Register Contents to Accumulator

Encoding: 0 1 1 0 1 r r r 68H-6FH

Description: The contents of register 'r' are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + (Rr)  
r = 0-7

Example: ADDREG: ADD A,R6  ;ADD REG 6 CONTENTS TO ACC

ADD A,@Ri Add Data Memory Contents to Accumulator

Encoding: 0 1 1 0 0 0 0 i 60H-61H

Description: The contents of the resident data memory location addressed by register 'i' bits 0-5** are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + ((Ri))  
i = 0-1

Example: ADDM: MOV R0, #01FH  ;MOVE '1F' HEX TO REG 0
ADD A, @R0  ;ADD VALUE OF LOCATION 31 TO ACC

ADD A,#data Add Immediate Data to Accumulator

Encoding: 0 0 0 0 0 0 1 1 d7 d6 d5 d4 d3 d2 d1 d0 03H

Description: This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.

Operation:  

Example: ADDID: ADD A,#ADDER:  ;ADD VALUE OF SYMBOL 'ADDER' TO ACC

ADDC A,Rr Add Carry and Register Contents to Accumulator

Encoding: 0 1 1 1 1 r r r 78H-7FH

Description: The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.

Operation: (A) ← (A) + (Rr) + (C)  
r = 0-7

Example: ADDRGC: ADDC A,R4  ;ADD CARRY AND REG 4 CONTENTS TO ACC

** 0-5 in 8048AH/8748H
0-6 in 8049AH/8749H
0-7 in 8050AH
MCS®-48 INSTRUCTION SET

**ADDC A, @Ri** Add Carry and Data Memory Contents to Accumulator

**Encoding:**

```
  0 1 1 1 0 0 0 i
```

**Description:**

The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'i' bits 0–5** are added to the accumulator. Carry is affected.

**Operation:**

\[(A) \leftarrow (A) + ((Ri)) + (C)\]

**Example:**

```
ADDCMC: MOV R1,#40
         ADDCA,@R1
```

; MOVE '40' DEC TO REG 1
; ADD CARRY AND LOCATION 40
; CONTENTS TO ACC

**ADDC A, @data** Add Carry and Immediate Data to Accumulator

**Encoding:**

```
  0 0 0 1 0 0 1 1
```

**Description:**

This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

**Operation:**

\[(A) \leftarrow (A) + \text{data} + (C)\]

**Example:**

```
ADDC A,#225
```

; ADD CARRY AND '225' DEC
; TO ACC

**ANL A,Rr** Logical AND Accumulator with Register Mask

**Encoding:**

```
  0 1 0 1 1 r r r
```

**Description:**

Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

**Operation:**

\[(A) \leftarrow (A) \text{AND} (Rr)\]

**Example:**

```
ANDREG: ANL A,R3
```

; AND ACC CONTENTS WITH MASK
; IN REG 3

**ANL A, @Ri** Logical AND Accumulator with memory Mask

**Encoding:**

```
  0 1 0 1 0 0 0 i
```

**Description:**

Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'i' bits 0–5**.

**Operation:**

\[(A) \leftarrow (A) \text{AND} ((Ri))\]

**Example:**

```
ANDDM: MOV R0,#03FH
        ANL A, @R0
```

; MOVE '3F' HEX TO REG 0
; AND ACC CONTENTS WITH
; MASK IN LOCATION 63

**0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH**
### ANL A,#data  Logical AND Accumulator with Immediate Mask

| Encoding | \[0101|0011\] d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 | 53H |
|----------|-----------------------------------------------|-----|
| Description: | This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask. |
| Operation: | (A) ← (A) AND data |
| Examples: | ANL A,#0AFH ;'AND' ACC CONTENTS WORLD MASK 10101111 |
|           | ANL A,#3 + X/Y ;'AND' ACC CONTENTS WITH VALUE OF EXP ;'3 + XY/Y' |

### ANL BUS,#data*  Logical AND BUS with Immediate Mask

| Encoding | \[1001|1000\] d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 | 98H |
|----------|-----------------------------------------------|-----|
| Description: | This is a 2-cycle instruction. Data on the BUS port is logically ANDed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS, A' instruction. |
| Operation: | (BUS) ← (BUS) AND data |
| Example: | ANL BUS: ANL BUS,#MASK ;'AND' BUS CONTENTS WITH MASK EQUAL VALUE OF SYMBOL 'MASK' |

### ANL Pp,#data  Logical AND Port 1-2 with Immediate Mask

| Encoding | \[1001|10pp\] d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 | 99H-9AH |
|----------|-----------------------------------------------|-------|
| Description: | This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask. |
| Operation: | (Pp) ← (Pp) AND DATA p = 1-2 |
| Example: | ANL P2: ANL P2,#0F0H ;'AND' PORT 2 CONTENTS WITH MASK 'F0' HEX ;(CLEAR P20-23) |

*For use with internal program memory ONLY.
ANLD Pp,A  Logical AND Port 4-7 with Accumulator Mask

Encoding: 1 0 0 1 1 p p 9CH-9FH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.

Operation: (Pp) ← (Pp) AND (AO-3)  p = 4-7

Note: The mapping of port 'p' to opcode bits 0-1 is as follows:

<table>
<thead>
<tr>
<th>Port</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

Example: ANDP4: ANLD P4,A

; 'AND' PORT 4 CONTENTS
; WITH ACC BITS 0-3

CALL address Subroutine Call

Encoding: a10 a9 a8 1 0 1 0 0 a7 a6 a5 a4 a3 a2 a1 a0

Description: This is a 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack. The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 is determined by the most recent SEL MB instruction.

A CALL cannot begin in locations 2046-2047 or 4094-4095. Execution continues at the instruction following the CALL upon return from the subroutine.

Operation: ((SP)) ← (PC), (PSW4-7)
(SP) ← (SP) + 1
(PC8-10) ← (addr8-10)
(PC0-7) ← (addr0-7)
(PC11) ← DBF
**Example:** Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```assembly
MOV R0,#50 ; MOVE '50' DEC TO ADDRESS
BEGADD: MOV A,R1 ; MOVE CONTENTS OF REG 1 TO ACC
           ADD A,R2 ; ADD REG 2 TO ACC
           CALL SUBTOT ; CALL SUBROUTINE 'SUBTOT'
           ADDC A,R3 ; ADD REG 3 TO ACC
           ADDC A,R4 ; ADD REG 4 TO ACC
           CALL SUBTOT ; CALL SUBROUTINE 'SUBTOT'
           ADDC A,R5 ; ADD REG 5 TO ACC
           ADDC A,R6 ; ADD REG 6 TO ACC
           CALL SUBTOT ; CALL SUBROUTINE 'SUBTOT'
SUBTOT: MOV @R0,A ; MOVE CONTENTS OF ACC TO LOCATION ADDRESSED BY REG 0
           INC R0 ; INCREMENT REG 0
           RET ; RETURN TO MAIN PROGRAM
```

**CLR A Clear Accumulator**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>0 0 1 0 0 1 1 1</th>
<th>27H</th>
</tr>
</thead>
</table>

**Description:** The contents of the accumulator are cleared to zero.

**Operation:** \( A \leftarrow 0 \)

**CLR C Clear Carry Bit**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>1 0 0 1 0 1 1 1</th>
<th>97H</th>
</tr>
</thead>
</table>

**Description:** During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA instructions. This instruction resets the carry bit to zero.

**Operation:** \( C \leftarrow 0 \)

**CLR F1 Clear Flag 1**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>1 0 1 0 0 1 0 1</th>
<th>A5H</th>
</tr>
</thead>
</table>

**Description:** Flag 1 is cleared to zero.

**Operation:** \( (F1) \leftarrow 0 \)
### CLR F0  Clear Flag 0

**Encoding:** [1 0 0 0 0 1 0 1] 85H  
**Description:** Flag 0 is cleared to zero.  
**Operation:** (F0) ← 0

---

### CPL A  Complement Accumulator

**Encoding:** [0 0 1 1 0 1 1] 37H  
**Description:** The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.  
**Operation:** (A) ← NOT (A)  
**Example:** Assume accumulator contains 01101010.  
CPLA: CPL A ;ACC CONTENTS ARE COMPLEMENTED TO 10010101

---

### CPL C  Complement Carry Bit

**Encoding:** [1 0 1 0 0 1 1] A7H  
**Description:** The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.  
**Operation:** (C) ← NOT (C)  
**Example:** Set C to one; current setting is unknown.  
CTO1: CLR C ;C IS CLEARED TO ZERO  
CPL C ;C IS SET TO ONE

---

### CPL F0  Complement Flag 0

**Encoding:** [1 0 0 1 0 1 0 1] 95H  
**Description:** The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.  
**Operation:** F0 ← NOT (F0)

---

### CPL F1  Complement Flag 1

**Encoding:** [1 0 1 1 0 1 0 1] B5H  
**Description:** The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.  
**Operation:** (F1) ← NOT (F1)
MCS®-48 INSTRUCTION SET

DA A  Decimal Adjust Accumulator

Encoding: 0 1 0 1 0 1 1 1 57H

Description: The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

Example: Assume accumulator contains 10011011.
DA A ;ACC Adjusted to 00000001
C AC 7 4 3 0
0 0 1 0 0 1 1 0 1 1 ADD SIX TO BITS 0-7
0 0 0 0 0 0 1 1 0
0 1 1 0 1 0 0 0 0 1 ADD SIX TO BITS 4-7
0 1 1 0 OVERFLOW TO C

DEC A  Decrement Accumulator

Encoding: 0 0 0 0 0 1 1 1 0 7H

Description: The contents of the accumulator are decremented by one. The carry flag is not affected.

Example: Decrement contents of external data memory location 63.
MOV R0,#3FH ;MOVE '3F' HEX TO REG 0
MOVX A, @R0 ;MOVE CONTENTS OF LOCATION 63 TO ACC
DEC A ;DECREMENT ACC
MOVX @R0,A ;MOVE CONTENTS OF ACC TO LOCATION 63 IN EXPANDED MEMORY

DEC Rr  Decrement Register

Encoding: 1 1 0 0 1 r r r 8H-CFH

Description: The contents of working register 'r' are decremented by one.

Operation: (Rr) ← (Rr) -1

Example: DECR1: DEC R1 ;DECREMENT CONTENTS OF REG 1
**MCS®-48 INSTRUCTION SET**

**DIS I  External Interrupt**

*Encoding:*

| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |

**Description:**

External interrupts are disabled. A low signal on the interrupt input pin has no effect.

**DIS TCNTI  Disable Timer/Counter Interrupt**

*Encoding:*

| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

**Description:**

Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

**DJNZ R, address  Decrement Register and Test**

*Encoding:*

| 1 | 1 | 1 | 0 | 1 | r | r | r |

**Description:**

This is a 2-cycle instruction. Register 'r' is decremented, then tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

**Example:**

\[(Rr) \rightarrow (Rr) - 1\]

If Rr not 0

\[(PC0-7) \rightarrow addr\]

Note: A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

**Example:**

Increment values in data memory locations 50–54.

- MOV R0,#50 ;MOVE '50' DEC TO ADDRESS
- MOV R3,#5 ;MOVE '5' DEC TO COUNTER
- INCRT: INC @R0 ;INCREMENT CONTENTS OF LOCATION ADDRESSED BY REG 0
- INC R0 ;INCREMENT ADDRESS IN REG 0
- DJNZ R3, INCRT ;DECREMENT REG 3 — JUMP TO 'INCRT' IF REG 3 NONZERO
- NEXT — ;'NEXT' ROUTINE EXECUTED IF R3 IS ZERO
MCS®-48 INSTRUCTION SET

EN I  Enable External Interrupt

Encoding: 0 0 0 0 0 1 0 1 05H
Description: External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.

EN TCNTI  Enable Timer/Counter Interrupt

Encoding: 0 0 1 0 0 1 0 1 25H
Description: Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.

ENT0 CLK  Enable Clock Output

Encoding: 0 1 1 1 0 1 0 1 75H
Description: The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.
Example: EMTST0: ENT0 CLK ; ENABLE T0 AS CLOCK OUTPUT

IN A,Pp  Input Port or Data to Accumulator

Encoding: 0 0 0 0 1 0 p p 09H-0AH
Description: This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.
Operation: (A) ← (Pp) p = 1–2
INP12: IN A,P1 ; INPUT PORT 1 CONTENTS TO ACC
MOV R6,A ; MOVE ACC CONTENTS TO REG 6
IN A,P2 ; INPUT PORT 2 CONTENTS TO ACC
MOV R7,A ; MOVE ACC CONTENTS TO REG 7

INC A  Increment Accumulator

Encoding: 0 0 0 1 0 1 1 1 17H
Description: The contents of the accumulator are incremented by one. Carry is not affected.
Operation: (A) ← (A) +1
Example: Increment contents of location 100 in external data memory.
INCA: MOV R0,#100 ;MOVE '100' DEC TO ADDRESS REG 0
MOVX A,@R0 ;MOVE CONTENTS OF LOCATION: 100 TO ACC
INC A ;INCREMENT A
MOVX @R0,A ;MOVE ACC CONTENTS TO LOCATION 101

INC Rr Increment Register

Encoding: 0 0 0 1 1 r r r 18H-1FH
Description: The contents of working register 'r' are incremented by one.
Operation: (Rr) ← (Rr) + 1  r = 0-7
Example: INCR0: INC R0 ;INCREMENT CONTENTS OF REG 0

INC @Ri Increment Data Memory Location

Encoding: 0 0 0 1 0 0 0 i 10H-11H
Description: The contents of the resident data memory location addressed by register 'i' bits 0-5** are incremented by one.
Operation: ((Ri)) ← ((Ri)) + 1  i = 0-1
Example: INCDM: MOV R1,#03FH INC @R1 ;MOVE ONES TO REG 1 ;INCREMENT LOCATION 63

INS A,BUS* Strobed Input of BUS Data to Accumulator

Encoding: 0 0 0 0 1 0 0 0 08H
Description: This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the RD pulse is dropped. (Refer to section on programming memory expansion for details.)
Operation: (A) ← (BUS)
Example: INPBUS: INS A,BUS ;INPUT BUS CONTENTS TO ACC

* For use with internal program memory ONLY.
** 0-5 in 8048AH/8748H
0-6 in 8049AH/8749H
0-7 in 8050AH
## MCS®-48 INSTRUCTION SET

### JBb address  Jump If Accumulator Bit Is Set

**Description:** This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

**Operation:**
- \((PC_{0-7}) \leftarrow addr\)
- \((PC) = (PC) + 2\)

**Example:**
- **JB4IS1:** JB4 NEXT
  - ;JUMP TO 'NEXT' ROUTINE
  - ;IF ACC BIT 4 = 1

**Encoding:**
- \[
  \begin{array}{cccccccc}
  b_2 & b_1 & b_0 & 0 & 0 & 1 & 0 & \text{Hex Op Code} \\
  \hline
  0 & 12 & 1 & 32 & 2 & 52 & 3 & 72 \\
  4 & 92 & 5 & B2 & 6 & D2 & 7 & F2 \\
  \end{array}
  \]

### JC address  Jump If Carry Is Set

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

**Operation:**
- \((PC_{0-7}) \leftarrow addr\)
- \((PC) = (PC) + 2\)

**Example:**
- **JC1:** JC OVFLOW
  - ;JUMP TO 'OVFLOW' ROUTINE
  - ;IF C = 1

**Encoding:**
- \[
  \begin{array}{cccccccc}
  1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & F6H \\
  \end{array}
  \]

### JF0 address  Jump If Flag 0 Is Set

**Description:** This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

**Operation:**
- \((PC_{0-7}) \leftarrow addr\)
- \((PC) = (PC) + 2\)

**Example:**
- **JF0IS1:** JF0 TOTAL
  - ;JUMP TO 'TOTAL' ROUTINE IF F0 = 1

**Encoding:**
- \[
  \begin{array}{cccccccc}
  1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & B6H \\
  \end{array}
  \]
### JF1 address
Jump If Flag 1 Is Set

**Encoding:**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>01110110</td>
<td>This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.</td>
<td>(PC) = (PC + 2) If F1 = 0</td>
<td>JF1S1: JF1 FILBUF:JUMP TO 'FILBUF' ROUTINE IF F1 = 1</td>
</tr>
</tbody>
</table>

**Example:**

```
JF1S1: JF1 FILBUF
;JUMP TO 'FILBUF'
;ROUTINE IF F1 = 1
```

### JMP address
Direct Jump within 2K Block

**Encoding:**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10a9a80100</td>
<td>This is a 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.</td>
<td>(PC8-10) = addr 8-10 If F1 = 1</td>
<td>JMP SUBTOT: JUMP TO SUBROUTINE 'SUBTOT' JUMP TO INSTRUCTION SIX LOCATIONS BEFORE CURRENT LOCATION JUMP TO ADDRESS '2F' HEX</td>
</tr>
</tbody>
</table>

**Example:**

```
JMP SUBTOT
JMP $-6
JMP 2FH
```

### JMPP @A
Indirect Jump within Page

**Encoding:**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>10110011</td>
<td>This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0-7).</td>
<td></td>
</tr>
</tbody>
</table>
MCS®-48 INSTRUCTION SET

**Operation:** \((PC_{0-7}) \leftarrow ((A))\)

**Example:** Assume accumulator contains 0FH.

JMPPAG: JMPP @A ;JUMP TO ADDRESS STORED IN LOCATION 15 IN CURRENT PAGE

### JNC address Jump If Carry Is Not Set

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>(1\ 1\ 1\ 0\ 0\ 1\ 1\ 0)</th>
<th>(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)</th>
<th>E6H</th>
</tr>
</thead>
</table>

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

**Operation:**
- \((PC_{0-7}) \leftarrow addr\) If \(C = 0\)
- \((PC) = (PC) + 2\) If \(C = 1\)

**Example:** JC0: JNC NOVFLO ;JUMP TO 'NOVFLO' ROUTINE ;IF C = 0

### JNI address Jump If Interrupt Input Is Low

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>(1\ 0\ 0\ 0\ 0\ 1\ 1\ 0)</th>
<th>(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)</th>
<th>86H</th>
</tr>
</thead>
</table>

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (= 0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

**Operation:**
- \((PC_{0-7}) \leftarrow addr\) If \(I = 0\)
- \((PC) = (PC) + 2\) If \(I = 1\)

**Example:** LOC 3: JNI EXTINT ;JUMP TO 'EXTINT' ROUTINE ;IF I = 0

### JNT0 address Jump If Test 0 is Low

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>(0\ 0\ 1\ 0\ 0\ 1\ 1\ 0)</th>
<th>(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)</th>
<th>26H</th>
</tr>
</thead>
</table>

**Description:** This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low.

**Operation:**
- \((PC_{0-7}) \leftarrow addr\) If \(T0 = 0\)
- \((PC) = (PC) + 2\) If \(T0 = 1\)

**Example:** JT0LOW: JNT0 60 ;JUMP TO LOCATION 60 DEC ;IF T0 = 0
MCS®-48 INSTRUCTION SET

JNT1 address  Jump If Test 1 Is Low

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>0 1 0 0 0 1 1 0</th>
<th>a7 a6 a5 a4 a3 a2 a1 a0</th>
<th>46H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(PC0-7) ← addr</td>
<td>If T1 = 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(PC) = (PC) + 2</td>
<td>If T1 = 1</td>
<td></td>
</tr>
</tbody>
</table>

JNZ Address  Jump If Accumulator Is Not Zero

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>1 0 0 1 0 1 1 0</th>
<th>a7 a6 a5 a4 a3 a2 a1 a0</th>
<th>96H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(PC0-7) ← addr</td>
<td>If A ≠ 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(PC) = (PC) + 2</td>
<td>If A = 0</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>JACCNO: JNZ 0ABH</td>
<td>;JUMP TO LOCATION ‘AB’ HEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>;IF ACC VALUE IS NONZERO</td>
<td></td>
</tr>
</tbody>
</table>

JTF address  Jump If Timer Flag Is Set

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>0 0 0 1 0 1 1 0</th>
<th>a7 a6 a5 a4 a3 a2 a1 a0</th>
<th>16H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(PC0-7) ← addr</td>
<td>If TF = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(PC) = (PC) + 2</td>
<td>If TF = 0</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>JTF1: JTF TIMER</td>
<td>;JUMP TO ‘TIMER’ ROUTINE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>;IF TF = 1</td>
<td></td>
</tr>
</tbody>
</table>

JT0 address  Jump If Test 0 Is High

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>0 0 1 1 0 1 1 0</th>
<th>a7 a6 a5 a4 a3 a2 a1 a0</th>
<th>36H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (= 1).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(PC0-7) ← addr</td>
<td>If T0 = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(PC) = (PC) + 2</td>
<td>If T0 = 0</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>JT0HI: JT0 53</td>
<td>;JUMP TO LOCATION 53 DEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>;IF T0 = 1</td>
<td></td>
</tr>
</tbody>
</table>
### JT1 address
#### Jump If Test 1 Is High

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>0101 0110</th>
</tr>
</thead>
<tbody>
<tr>
<td>a7 a6 a5 a4 a3 a2 a1 a0</td>
<td>56H</td>
</tr>
</tbody>
</table>

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (= 1).

**Operation:**
- \((PC_0-7) \leftarrow addr\)
- \((PC) = (PC) + 2\)

**Example:**
- JT1HI: JT1 COUNT
  - ; JUMP TO 'COUNT' ROUTINE
  - ; IF T1 = 1

### JZ address
#### Jump If Accumulator Is Zero

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>1100 0110</th>
</tr>
</thead>
<tbody>
<tr>
<td>a7 a6 a5 a4 a3 a2 a1 a0</td>
<td>C6H</td>
</tr>
</tbody>
</table>

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

**Operation:**
- \((PC_0-7) \leftarrow addr\)
- \((PC) = (PC) + 2\)

**Example:**
- JACCO: JZ 0A3H
  - ; JUMP TO LOCATION 'A3' HEX
  - ; IF ACC VALUE IS ZERO

### MOV A,#data
#### Move Immediate Data to Accumulator

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>0010 0011</th>
</tr>
</thead>
<tbody>
<tr>
<td>a7 a6 a5 a4 a3 a2 a1 a0</td>
<td>23H</td>
</tr>
</tbody>
</table>

**Description:** This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

**Operation:** \((A) \leftarrow data\)

**Example:**
- MOV A,#0A3H
  - ; MOVE 'A3' HEX TO ACC

### MOV A,PSW
#### Move PSW Contents to Accumulator

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>1100 0111</th>
</tr>
</thead>
<tbody>
<tr>
<td>a7 a6 a5 a4 a3 a2 a1 a0</td>
<td>C7H</td>
</tr>
</tbody>
</table>

**Description:** The contents of the program status word are moved to the accumulator.

**Operation:** \((A) \leftarrow (PSW)\)

**Example:**
- Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.
  - BSCHK: MOV A,PSW
  - JB4 RB1SET ; MOVE PSW CONTENTS TO ACC
  - JUMP TO 'RB1SET' IF ACC BIT 4 = 1
### MCS®-48 INSTRUCTION SET

#### MOV A,Rₙ  Move Register Contents to Accumulator

- **Encoding:** \[1111\ 1\ r\ r\]  \(F8H-FFH\)
- **Description:** 8-bits of data are removed from working register \(r\) into the accumulator.
- **Operation:** \((A) \leftarrow (R_r)\)  \(r = 0-7\)
- **Example:** MAR: MOV A,R3 ;MOVE CONTENTS OF REG 3 TO ACC

#### MOV A,@Rᵢ  Move Data Memory Contents to Accumulator

- **Encoding:** \[1111\ 00\ 0\ i\]  \(F0H-F1H\)
- **Description:** The contents of the resident data memory location addressed by bits 0-5** of register \(i\) are moved to the accumulator. Register \(i\) contents are unaffected.
- **Operation:** \((A) \leftarrow ((R_i)\)\)  \(i = 0-1\)
- **Example:** Assume R1 contains 00110110. MADM: MOV A,@R1 ;MOVE CONTENTS OF DATA MEM ;LOCATION 54 TO ACC

#### MOV A,T  Move Timer/Counter Contents to Accumulator

- **Encoding:** \[0100\ 0010\]  \(42H\)
- **Description:** The contents of the timer/event-counter register are moved to the accumulator.
- **Operation:** \((A) \leftarrow (T)\)
- **Example:** Jump to “EXIT” routine when timer reaches ‘64’, that is, when bit 6 set—assuming initialization 64, TIMCHK: MOV A,T ;MOVE TIMER CONTENTS TO ACC JB6 EXIT ;JUMP TO ‘EXIT’ IF ACC BIT 6 = 1

#### MOV PSW,A  Move Accumulator Contents to PSW

- **Encoding:** \[1101\ 0111\]  \(D7H\)
- **Description:** The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.
- **Operation:** \((PSW) \leftarrow (A)\)
- **Example:** Move up stack pointer by two memory locations, that is, increment the pointer by one.
  INC PTR: MOV A,PSW ;MOVE PSW CONTENTS TO ACC INC A ;INCREMENT ACC BY ONE MOV PSW,A ;MOVE ACC CONTENTS TO PSW

** 0-5 in 8048AH/8748H  
0-6 in 8049AH/8749H  
0-7 in 8050AH
## MOV R<sub>r</sub>,A  Move Accumulator Contents to Register

<table>
<thead>
<tr>
<th>Encoding</th>
<th>1 0 1 0 1 r r r</th>
<th>A8H–AFH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The contents of the accumulator are moved to register 'r'.</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>(R&lt;sub&gt;r&lt;/sub&gt;) ← (A)</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>MRA: MOV R0,A</td>
<td>;MOVE CONTENTS OF ACC TO REG 0</td>
</tr>
</tbody>
</table>

## MOV R<sub>r</sub>,#data  Move Immediate Data to Register

<table>
<thead>
<tr>
<th>Encoding</th>
<th>1 0 1 1 1 r&lt;sub&gt;2&lt;/sub&gt; r&lt;sub&gt;1&lt;/sub&gt; r&lt;sub&gt;0&lt;/sub&gt;</th>
<th>d&lt;sub&gt;7&lt;/sub&gt; d&lt;sub&gt;6&lt;/sub&gt; d&lt;sub&gt;5&lt;/sub&gt; d&lt;sub&gt;4&lt;/sub&gt; d&lt;sub&gt;3&lt;/sub&gt; d&lt;sub&gt;2&lt;/sub&gt; d&lt;sub&gt;1&lt;/sub&gt; d&lt;sub&gt;0&lt;/sub&gt;</th>
<th>B8H–BFH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>(R&lt;sub&gt;r&lt;/sub&gt;) ← data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Examples</td>
<td>MIR4: MOV R4,#HEXTEN</td>
<td>;THE VALUE OF THE SYMBOL 'HEXTEN' IS MOVED INTO REG 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIR 5: MOV R5,#PI*(R*R)</td>
<td>;THE VALUE OF THE EXPRESSION 'PI*(R*R)' IS MOVED INTO REG 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIR 6: MOV R6,#0ADH</td>
<td>;'AD' HEX IS MOVED INTO REG 6</td>
<td></td>
</tr>
</tbody>
</table>

## MOV @ R<sub>i</sub>,A  Move Accumulator Contents to Data Memory

<table>
<thead>
<tr>
<th>Encoding</th>
<th>1 0 1 0 0 0 0 i</th>
<th>A0H–A1H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5** of register 'i'. Register 'i' contents are unaffected.</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>((R&lt;sub&gt;i&lt;/sub&gt;)) ← (A)</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>Assume R0 contains 00000111. MDMA: MOV @R0,A</td>
<td>;MOVE CONTENTS OF ACC TO LOCATION 7 (REG 7)</td>
</tr>
</tbody>
</table>

## MOV @ R<sub>i</sub>,#data  Move Immediate Data to Data memory

<table>
<thead>
<tr>
<th>Encoding</th>
<th>1 0 1 1 0 0 0 i</th>
<th>d&lt;sub&gt;7&lt;/sub&gt; d&lt;sub&gt;6&lt;/sub&gt; d&lt;sub&gt;5&lt;/sub&gt; d&lt;sub&gt;4&lt;/sub&gt; d&lt;sub&gt;3&lt;/sub&gt; d&lt;sub&gt;2&lt;/sub&gt; d&lt;sub&gt;1&lt;/sub&gt; d&lt;sub&gt;0&lt;/sub&gt;</th>
<th>B0H–B1H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'i', bits 0-5**.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>((R&lt;sub&gt;i&lt;/sub&gt;)) ← data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Examples</td>
<td>Move the hexadecimal value AC3F to locations 62–63. MIDM: MOV R0,#62</td>
<td>;MOVE '62' DEC TO ADDR REG 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOV @R0,#0ACH.</td>
<td>;MOVE 'AC' HEX TO LOCATION 62</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INC R0</td>
<td>;INCREMENT REG 0 to '63'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOV @R0,#3FH</td>
<td>;MOVE '3F' HEX TO LOCATION 63</td>
<td></td>
</tr>
</tbody>
</table>

**0–5 in 8048AH/8748H  
0–6 in 8049AH/8749H  
0–7 in 8050AH
MOV T,A  Move Accumulator Contents to Timer/Counter

**Encoding:**

```
0 1 1 0 0 0 1 0
```

**Description:** The contents of the accumulator are moved to the timer/event-counter register.

**Operation:** \( (T) \leftarrow (A) \)

**Example:** Initialize and start event counter.

```
INITEC: CLR A ;CLEAR ACC TO ZEROS
MOV T,A ;MOVE ZEROS TO EVENT COUNTER
START CNT ;START COUNTER
```

MOVD A,Pp  Move Port 4-7 Data to Accumulator

**Encoding:**

```
0 0 0 0 1 1 p p
```

**Description:** This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

**Operation:**

\[
(0-3) \leftarrow (Pp) \quad \text{if } p = 4-7 \\
(4-7) \leftarrow 0
\]

**Example:** INPPT5: MOVD A,P5 ;MOVE PORT 5 DATA TO ACC ;BITS 0-3, ZERO ACC BITS 4-7

MOVD Pp,A  Move Accumulator Data to Port 4-7

**Encoding:**

```
0 0 1 1 1 1 p p
```

**Description:** This is a 2-cycle instruction. Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

**Operation:** \( (Pp) \leftarrow (A_{0-3}) \) \quad P = 4-7

**Example:** OUTP45: MOVD P4,A ;MOVE ACC BITS 0-3 TO PORT 4 
SWAP A ;EXCHANGE ACC BITS 0-3 and 4-7
MOVD P5,A ;MOVE ACC BITS 0-3 TO PORT 5
MCS®-48 INSTRUCTION SET

MOVP A, @A  Move Current Page Data to Accumulator

**Encoding:**

\[ \begin{array}{cccccc} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{array} \]

A3H

**Description:**
The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored following this operation.

**Operation:**

\[
(PC_{0-7}) \leftarrow (A) \\
(A) \leftarrow ((PC))
\]

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the following page.

**Example:**

MOV128: MOV A,#128 ; MOVE '128' DEC TO ACC
MOVP A, @A ; CONTENTS OF 129th LOCATION IN CURRENT PAGE ARE MOVED TO ACC

MOVP3 A, @A  Move Page 3 Data to Accumulator

**Encoding:**

\[ \begin{array}{cccccc} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{array} \]

E3H

**Description:**
This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

**Operation:**

\[
(PC_{0-7}) \leftarrow (A) \\
(PC_{8-11}) \leftarrow 0011 \\
(A) \leftarrow ((PC))
\]

**Example:**

Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

**Example:**

TABSCH: MOV A,#0B8H ; MOVE 'B8' HEX TO ACC (10111000)
ANL A,#7FH ; LOGICAL AND ACC TO MASK BIT ; 7 (00111000)
MOVP3 A, @A ; MOVE CONTENTS OF LOCATION '38' ; HEX IN PAGE 3 TO ACC (ASCII '8')

Access contents of location in page 3 labelled TAB1.
Assume current program location is not in page 3.

TABSCH: MOV A,#LOW TAB 1 ; ISOLATE BITS 0-7 OF LABEL ; ADDRESS VALUE
MOVP3 A, @A ; MOVE CONTENTS OF PAGE 3 ; LOCATION LABELED 'TAB1' TO ACC
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Encoding</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVX A,@Ri</td>
<td>Move External-Data-Memory Contents to Accumulator</td>
<td>1000 000 i 80H–81H</td>
<td>(A) ← ((Ri)) i = 0–1</td>
<td>MAXDM: MOVX A,@R1 ;MOVE CONTENTS OF LOCATION 118 TO ACC</td>
</tr>
<tr>
<td>MOVX @Ri,A</td>
<td>Move Accumulator Contents to External Data Memory</td>
<td>1001 000 i 90H–91H</td>
<td>((Ri)) ← A i = 0–1</td>
<td>MXDMA: MOVX @R0,A ;MOVE CONTENTS OF ACC TO LOCATION 199 IN EXPANDED DATA MEMORY</td>
</tr>
<tr>
<td>NOP</td>
<td>The NOP Instruction</td>
<td>0000 000 0 00H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORL A,Rr</td>
<td>Logical OR Accumulator With Register Mask</td>
<td>0100 1rr r 48H–4FH</td>
<td>(A) ← (A) OR (Rr) r = 0–7</td>
<td>ORREG: ORL A,R4 ;'OR' ACC CONTENTS WITH MASK IN REG 4</td>
</tr>
</tbody>
</table>
### MCS®-48 INSTRUCTION SET

**ORL A, @Ri** Logical OR Accumulator With Memory Mask

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>0100 000i</th>
<th>40H–41H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register “i”, bits 0–5**.</td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(A)←(A) OR ((Ri)) i = 0–1</td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>ORDM: MOV R0, #3FH; MOVE ‘3F’ HEX TO REG 0 ORL A, @R0; ‘OR’ AC CONTENTS WITH MASK IN LOCATION 63</td>
<td></td>
</tr>
</tbody>
</table>

**ORL A, #data** Logical OR Accumulator With Immediate Mask

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>0100 0011</th>
<th>d7 d6 d5 d4 d3 d2 d1 d0</th>
<th>43H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(A)←(A) OR data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>ORID: ORL A, #’X’; ‘OR’ ACC CONTENTS WITH MASK; 01011000 (ASCII VALUE OF ‘X’)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ORL BUS, #data** Logical OR BUS With Immediate Mask

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>1000 1000</th>
<th>d7 d6 d5 d4 d3 d2 d1 d0</th>
<th>88H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification on an ‘OUTL BUS, A’ instruction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(BUS)←(BUS) OR data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>ORBUS: ORL BUS, #HEXMSK; ‘OR’ BUS CONTENTS WITH MASK; EQUAL VALUE OF SYMBOL ‘HEXMSK’</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ORL Pp, #data** Logical OR Port 1 or 2 With Immediate Mask

<table>
<thead>
<tr>
<th>Encoding:</th>
<th>1000 10pp</th>
<th>d7 d6 d5 d4 d3 d2 d1 d0</th>
<th>89H–8AH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>This is a 2-cycle instruction. Data on port ‘p’ is logically ORed with an immediately-specified mask.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation:</td>
<td>(Pp)←(Pp) OR data p = 1–2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example:</td>
<td>ORP1: ORL P1, #0FFH; ‘OR’ PORT 1 CONTENTS WITH MASK; ‘FF’ HEX (SET PORT 1 TO ALL ONES)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* For use with internal program memory ONLY. ** 0–5 in 8048AH/8748H 0–6 in 8049AH/8749H 0–7 in 8050AH
MC8S®-48 INSTRUCTION SET

ORLD Pp,A Logical OR Port 4-7 With Accumulator Mask

Encoding: 1 0 0 0 1 1 p p 8CH-8FH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

Operation: (Pp) ← (Pp) OR (A0-3) p = 4-7

Example: ORP7: ORLD P7,A ; 'OR' PORT 7 CONTENTS WITH ACC ; BITS 0-3

OUTL BUS,A* Output Accumulator Data to BUS

Encoding: 0 0 0 0 0 0 1 0 02H

Description: This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.

Operation: (BUS) ← (A)

Example: OUTLBP: OUTL BUS, A ; OUTPUT ACC CONTENTS TO BUS

OUTL Pp,A Output Accumulator Data to Port 1 or 2

Encoding: 0 0 1 1 1 0 p p 39H-3AH

Description: This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.

Operation: (Pp) ← (A) p = 1-2

Example: OUTLP: MOV A,R7 ; MOVE REG 7 CONTENTS TO ACC
OUTL P2,A ; OUTPUT ACC CONTENTS TO PORT 2
MOV A, R6 ; MOVE REG 6 CONTENTS TO ACC
OUTL P1,A ; OUTPUT ACC CONTENTS TO PORT 1

* For use with internal program memory ONLY.
RET  Return Without PSW Restore

**Encoding:** 1 0 0 0 0 0 1 1 83H

**Description:** This is a 2-cycle instruction. The stack pointer (PSW bits 0–2) is decremented. The program counter is then restored from the stack. PSW bits 4–7 are not restored.

**Operation:**

\[
\begin{align*}
\text{SP} & \leftarrow \text{SP} - 1 \\
\text{PC} & \leftarrow \text{SP}
\end{align*}
\]

RETR  Return with PSW Restore

**Encoding:** 1 0 0 1 0 0 1 1 93H

**Description:** This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4–7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine by resetting the Interrupt in Progress flip-flop.

**Operation:**

\[
\begin{align*}
\text{SP} & \leftarrow \text{SP} - 1 \\
\text{PC} & \leftarrow \text{SP} \\
\text{PSW 4-7} & \leftarrow \text{SP}
\end{align*}
\]
### MCS-48 INSTRUCTION SET

#### RL A Rotate Left without Carry

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
</table>
| 11 1 0 0 1 1 1 1 E7H | The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position. | (An + 1) → (An)  
(A0) → (A7) | Assume accumulator contains 10110001.  
RLNC: RL A ; NEW ACC CONTENTS ARE 01100011 |

#### RLC A Rotate Left through Carry

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
</table>
| 11 1 1 0 1 1 1 F7H | The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position. | (An + 1) → (An)  
(A0) → (C)  
(C) → (A7) | Assume accumulator contains a 'signed' number; isolate sign without changing value.  
RLTC: CLR C ; CLEAR CARRY TO ZERO  
RLC A ; ROTATE ACC LEFT, SIGN  
RR A ; ROTATE ACC RIGHT — VALUE  
(CARRY UNCHANGED, BIT 7 IS ZERO |

#### RR A Rotate Right without Carry

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
</table>
| 0 1 1 0 0 1 1 1 77H | The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position. | (An) → (An + 1)  
(A7) → (A0) | Assume accumulator contains 10110001.  
RRNC: RR A ; NEW ACC CONTENTS ARE 11011000 |
### RRC A  Rotate Right through Carry

**Encoding:** 0110 0111  67H

**Description:** The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

**Operation:**
- \((A_n) \leftarrow (A_n + 1)\)
- \((A_7) \leftarrow (C)\)
- \((C) \leftarrow (A_0)\)

**Example:** Assume carry is not set and accumulator contains 10110001.
- RRTC: RRC A
- ; CARRY IS SET AND ACC
- ; CONTENTS 01011000

### SEL MB0  Select Memory Bank 0

**Encoding:** 1110 0101  E5H

**Description:** PC bit 11 is set to zero on next JMP or CALL instruction. All references to program memory addresses fall within the range 0-2047.

**Operation:** (DBF) \leftarrow 0

**Example:** Assume program counter contains 834 Hex.
- SEL MB0
- ; SELECT MEMORY BANK 0
- JMP $+20
- ; JUMP TO LOCATION 58 HEX

### SEL MB1  Select Memory Bank 1

**Encoding:** 1111 0101  F5H

**Description:** PC bit 11 is set to one on next JMP or CALL instruction. All references to program memory addresses fall within the range 2048-4095.

**Operation:** (DBF) \leftarrow 1
**SEL RB0  Select Register Bank 0**

**Encoding:** 11000101 C5H  
**Description:** PSW bit 4 is set to zero. References to working registers 0–7 address data memory locations 0–7. This is the recommended setting for normal program execution.  
**Operation:** (BS) ← 0

**SEL RB1  Select Register Bank 1**

**Encoding:** 11010101 D5H  
**Description:** PSW bit 4 is set to one. References to working registers 0–7 address data memory locations 24–31. This is the recommended setting for interrupt service routines, since locations 0–7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.  
**Operation:** (BS) ← 1  
**Example:** Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.  
**Operation:** LOC3: JNI INIT ;JUMP TO ROUTINE 'INIT' IF INTERRUPT INPUT IS ZERO  
INIT: MOV R7,A ;MOVE ACC CONTENTS TO LOCATION 7  
SEL RB1 ;SELECT REG BANK 1  
MOV R7,#0FAH ;MOVE 'FA' HEX TO LOCATION 31

**STOP TCNT  Stop Timer/Event-Counter**

**Encoding:** 01100101 65H  
**Description:** This instruction is used to stop both time accumulation and event counting.
Example: Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

START: DIS TCNTI
;DISABLE TIMER INTERRUPT
CLR A
;CLEAR ACC TO ZEROS
MOV T,A
;MOVE ZEROS TO TIMER
MOV R7,A
;MOVE ZEROS TO REG 7
STR T
;START TIMER

MAIN: JTF COUNT
;IF TF = 1 AND CLEAR TIMER FLAG
JMP MAIN
;CLOSE LOOP

COUNT: INC R7
;INCREMENT REG 7
MOV A,R7
;MOVE REG 7 CONTENTS TO ACC
JB3 INT
;JUMP TO ROUTINE 'INT' IF ACC
;BIT 3 IS SET (REG 7 = 8)
JMP MAIN
;OTHERWISE RETURN TO ROUTINE

INT: STOP TCNT
;STOP TIMER
JMP 7H
;JUMP TO LOCATION 7 (TIMER)
;INTERRUPT ROUTINE

STR T CNT  Start Event Counter

<table>
<thead>
<tr>
<th>Encoding</th>
<th>01000101</th>
<th>45H</th>
</tr>
</thead>
</table>

Description: The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

Example: Initialize and start event counter. Assume overflow is desired with first T1 input.

STARTC: EN TCNTI
;ENABLE COUNTER INTERRUPT
MOV A,#0FFH
;MOVE 'FF'HEX (ONES) TO ACC
MOV T,A
;MOVES ONES TO COUNTER
STR T CNT
;ENABLE T1 AS COUNTER
;INPUT AND START

INT: STOP TCNT
;STOP TIMER
JMP 7H
;JUMP TO LOCATION 7 (TIMER)
;INTERRUPT ROUTINE
STRT T  Start Timer

Encoding: 0 1 0 1 0 1 0 1 55H

Description: Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

Example: Initialize and start timer.

STARTT: CLR A  ;CLEAR ACC TO ZEROS
          MOV T,A    ;MOVE ZEROS TO TIMER
          EN TCNTI   ;ENABLE TIMER INTERRUPT
          STRT T     ;START TIMER

SWAP A  Swap Nibbles within Accumulator

Encoding: 0 1 0 0 0 1 1 1 47H

Description: Bits 0–3 of the accumulator are swapped with bits 4–7 of the accumulator.

Example: Pack bits 0–3 of locations 50–51 into location 50.

PCKDIG: MOV R0, #50  ;MOVE '50' DEC TO REG 0
          MOV R1, #51  ;MOVE '51' DEC TO REG 1
          XCHD A,@R0  ;EXCHANGE BITS 0–3 OF ACC
                        ;AND LOCATION 50
          SWAP A    ;SWAP BITS 0–3 AND 4–7 OF ACC
          XCHD A,@R1 ;EXCHANGE BITS 0–3 OF ACC AND
                        ;LOCATION 51
          MOV @R0,A  ;MOVE CONTENTS OF ACC TO
                        ;LOCATION 50

XCH A,Rr  Exchange Accumulator-Register Contents

Encoding: 0 0 1 0 1 r r r 28H-2FH

Description: The contents of the accumulator and the contents of working register 'r' are exchanged.

Operation: (A) ⇨ (Rr)  r = 0–7

Example: Move PSW contents to Reg 7 without losing accumulator contents.

XCHAR7: XCH A,R7  ;EXCHANGE CONTENTS OF REG 7
                   ;AND ACC
          MOV A, PSW ;MOVE PSW CONTENTS TO ACC
          XCH A,R7   ;EXCHANGE CONTENTS OF REG 7
                   ;AND ACC AGAIN
### XCH A,@R<sub>i</sub>  
**Exchange Accumulator and Data Memory Contents**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| 0 0 1 0 0 0 i      | The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5** of register ‘i’ are exchanged. Register ‘i’ contents are unaffected. | DEC52: MOV R0,#52  
XCH A,@R0  
DEC A  
XCH A,@R0 |
| 20H-21H           |                                                                           |         |

**Operation:**  
(A) \(\leftrightarrow\) ((R<i>)) \(i = 0-1\)

**Example:**  
Decrement contents of location 52.  
DEC52: MOV R0,#52  
XCH A,@R0  
DEC A  
XCH A,@R0  

### XCHD A,@R<sub>i</sub>  
**Exchange Accumulator and Data Memory 4-Bit Data**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| 0 0 1 1 0 0 i      | This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5** of register ‘i’. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register ‘i’ are unaffected. | XCHNIB: MOV R0,#23  
CLR A  
XCHD A,@R0  
XCHD A,@R0 |
| 30H-31H           |                                                                           |         |

**Operation:**  
(A<sub>0-3</sub>) \(\leftrightarrow\) ((R<i>-0-3>) \(i = 0-1\)

**Example:**  
Assume program counter contents have been stacked in locations 22-23.  
XCHNIB: MOV R0,#23  
CLR A  
XCHD A,@R0  
XCHD A,@R0  

### XRL A,R<sub>r</sub>  
**Logical XOR Accumulator With Register Mask**

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
</table>
| 1 1 0 1 1 r r r | Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register ‘r’. | XORREG: XRL A,R5  
;XOR' ACC CONTENTS WITH |
| D8H-DFH           |                                                                           |         |

**Operation:**  
(A) \(-\) (A) XOR (R<r>) \(r = 0-7\)

**Example:**  
XORREG: XRL A,R5  

**Notes:**  
** 0-5 in 8048AH/8748H  
0-6 in 8049AH/8749H  
0-7 in 8050AH
MCS®-48 INSTRUCTION SET

XRL A, @Ri  Logical XOR Accumulator With Memory Mask

Encoding: 1 1 0 1 0 0 0 i  D0H-D1H

Description: Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register ‘i’, bits 0–5.**

Operation: (A) ← (A) XOR ((Ri))

Example: XORDM: MOV R1,#20H
          XRL A, @R1

   i = 0–1

;MOVE ‘20’ HEX TO REG 1
;‘XOR’ ACC CONTENTS WITH MASK
;IN LOCATION 32

XRL A, #data  Logical XOR Accumulator With Immediate Mask

Encoding: 1 1 0 1 0 0 1 i  d7 d6 d5 d4 d3 d2 d1 d0  D3H

Description: This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

Operation: (A) ← (A) XOR data

Example: XORID: XOR A, #HEXTEN

   ;XOR CONTENTS OF ACC WITH MASK
   ;EQUAL VALUE OF SYMBOL ‘HEXTEN’

** 0–5 in 8048AH/8748H
   0–6 in 8049AH/8749H
   0–7 in 8050AH
8243
MCS®-48 INPUT/OUTPUT EXPANDER

- 0°C to 70°C Operation

Figure 1. 8243 Block Diagram

Figure 2. 8243 Pin Configuration
Functional Description

General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if VCC drops below 1V.

Write Modes

The device has three write modes. MOV D Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data. OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.

Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG</td>
<td>7</td>
<td>Clock Input A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.</td>
</tr>
<tr>
<td>CS</td>
<td>6</td>
<td>Chip Select Input A high on CS inhibits any change of output or internal status</td>
</tr>
<tr>
<td>P20-P23</td>
<td>11-8</td>
<td>Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.</td>
</tr>
<tr>
<td>GND</td>
<td>12</td>
<td>0 volt supply</td>
</tr>
<tr>
<td>P40-P43</td>
<td>2-5</td>
<td>Four (4) bit bi-directional I/O ports</td>
</tr>
<tr>
<td>P50-P53</td>
<td>1, 23-21</td>
<td>May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data.</td>
</tr>
<tr>
<td>P60-P63</td>
<td>20-17</td>
<td></td>
</tr>
<tr>
<td>P70-P73</td>
<td>13-16</td>
<td></td>
</tr>
<tr>
<td>VCC</td>
<td>24</td>
<td>+5 volt supply</td>
</tr>
</tbody>
</table>

Table 1. Pin Description

<table>
<thead>
<tr>
<th>Address Code</th>
<th>Instruction Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>P21 P20</td>
<td>P23 P22</td>
</tr>
<tr>
<td>0 0</td>
<td>Port 4</td>
</tr>
<tr>
<td>0 1</td>
<td>Port 5</td>
</tr>
<tr>
<td>1 0</td>
<td>Port 6</td>
</tr>
<tr>
<td>1 1</td>
<td>Port 7</td>
</tr>
<tr>
<td></td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>Write</td>
</tr>
<tr>
<td></td>
<td>ORLD</td>
</tr>
<tr>
<td></td>
<td>ANLD</td>
</tr>
</tbody>
</table>

Write Modes

The device has three write modes. MOV D Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data. OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.

Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ............ 0°C to 70°C
Storage Temperature .................... -65°C to +150°C
Voltage on Any Pin..................... -0.5 V to +7V
With Respect to Ground ................. 0.0 V
Power Dissipation ...................... 1 Watt

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5V ± 10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIH</td>
<td>Input High Voltage</td>
<td>2.0</td>
<td></td>
<td>VCC±0.5</td>
<td>V</td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage Ports 4-7</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 4.5 mA*</td>
<td></td>
</tr>
<tr>
<td>VOL2</td>
<td>Output Low Voltage Port 7</td>
<td>1</td>
<td>V</td>
<td>IOL = 20 mA</td>
<td></td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage</td>
<td>2.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIL1</td>
<td>Input Leakage Ports 4-7</td>
<td>-10</td>
<td>20</td>
<td>μA</td>
<td>Vin = VCC to OV</td>
</tr>
<tr>
<td>IIL2</td>
<td>Input Leakage Port 2, CS, PROG</td>
<td>-10</td>
<td>10</td>
<td>μA</td>
<td>Vin = VCC to OV</td>
</tr>
<tr>
<td>VOL3</td>
<td>Output Low Voltage Port 2</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 0.6 mA</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>VCC Supply Current</td>
<td>10</td>
<td>20</td>
<td>mA</td>
<td>Note 1</td>
</tr>
<tr>
<td>VOH2</td>
<td>Output Voltage Port 2</td>
<td>2.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOL</td>
<td>Sum of all IOL from 16 Outputs</td>
<td>72</td>
<td></td>
<td>4.5 mA Each Pin</td>
<td></td>
</tr>
</tbody>
</table>

*See following graph for additional sink current capability

A.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5V ± 10%)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>tA</td>
<td>Code Valid Before PROG</td>
<td>50</td>
<td>ns</td>
<td>80 pF Load</td>
</tr>
<tr>
<td>tB</td>
<td>Code Valid After PROG</td>
<td>60</td>
<td>ns</td>
<td>20 pF Load</td>
</tr>
<tr>
<td>tC</td>
<td>Data Valid Before PROG</td>
<td>200</td>
<td>ns</td>
<td>80 pF Load</td>
</tr>
<tr>
<td>tD</td>
<td>Data Valid After PROG</td>
<td>20</td>
<td>ns</td>
<td>20 pF Load</td>
</tr>
<tr>
<td>tH</td>
<td>Floating After PROG</td>
<td>0</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>tK</td>
<td>PROG Negative Pulse Width</td>
<td>700</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tCS</td>
<td>CS Valid Before/After PROG</td>
<td>50</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tPO</td>
<td>Ports 4-7 Valid Before PROG</td>
<td>700</td>
<td>ns</td>
<td>100 pF Load</td>
</tr>
<tr>
<td>tLP1</td>
<td>Ports 4-7 Valid After PROG</td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>tACC</td>
<td>Port 2 Valid Before PROG</td>
<td>650</td>
<td>80 pF Load</td>
<td></td>
</tr>
</tbody>
</table>

Note 1: ICC (-40°C to 85°C EXPRESS options) 15 mA typical/25 mA maximum.
WAVEFORMS

A.C. Testing: Inputs are driven at 2.4V for a logic '1' and 0.45V for a logic '0'. Output timing measurements are made at 2.0V for a logic '1' and 0.8V for a logic '0'.

---

WAVEFORMS

PROG

PORT 2

INSTRUCTION FLOAT DATA FLOAT

PORT 2

OUTPUT VALID

PORTS 4-7

PREVIOUS OUTPUT VALID

OUTPUT VALID

PORTS 4-7

INPUT VALID

CS
Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total IOL must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

\[ I_{OL} = 5 \times 1.6 \text{ mA} = 8 \text{ mA} \]
\[ I_{OL} = 60 \text{ mA from curve} \]
\# pins = 60 mA / 8 mA/pin = 7.5 = 7

In this case, 7 lines can sink 8 mA for a total of 56 mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads — 20 mA @ 1V (port 7 only)
8 loads — 4 mA @ .45V
6 loads — 3.2 mA @ .45V

Is this within the specified limits?

\[ I_{OL} = (2 \times 20) + (8 \times 4) + (6 \times 3.2) = 91.2 \text{ mA} \]

From the curve, for \( I_{OL} = 4 \text{ mA} \), \( I_{OL} = 93 \text{ mA} \).
Since 91.2 mA < 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating \( I_{OL} \), it is the largest current required @ .45V which determines the maximum allowable \( I_{OL} \).

NOTE: A10 to 50K pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.
**Figure 4. Expander Interface**

![Diagram](image)

**Figure 5. Output Expander Timing**

![Diagram](image)

**Figure 6. Using Multiple 8243's**

![Diagram](image)
HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- High Performance HMOS II
- Interval Timer/Event Counter
- Two Single Level Interrupts
- Single 5-Volt Supply
- Over 96 Instructions; 90% Single Byte
- Reduced Power Consumption
- Compatible with 8080/8085 Peripherals
- Easily Expandable Memory and I/O
- Up to 1.36 µSec Instruction Cycle
- All Instructions 1 or 2 cycles

The Intel MCS®-48 family are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The family contains 27 I/O lines, an 8-bit timer/counter, and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

To minimize development problems and provide maximum flexibility, a logically and functionally pin-compatible version of the ROM devices with UV-erasable user-programmable EPROM program memory is available with minor differences.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

<table>
<thead>
<tr>
<th>Device</th>
<th>Internal Memory</th>
<th>RAM Standby</th>
</tr>
</thead>
<tbody>
<tr>
<td>8050AH</td>
<td>4K × 8 ROM</td>
<td>yes</td>
</tr>
<tr>
<td>8049AH</td>
<td>2K × 8 ROM</td>
<td>yes</td>
</tr>
<tr>
<td>8048AH</td>
<td>1K × 8 ROM</td>
<td>yes</td>
</tr>
<tr>
<td>8040AHL</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>8039AHL</td>
<td>none</td>
<td>yes</td>
</tr>
<tr>
<td>8035AHL</td>
<td>none</td>
<td>yes</td>
</tr>
</tbody>
</table>

Figure 1. Block Diagram

Figure 2. Logic Symbol

Figure 3. Pin Configuration
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSS</td>
<td>20</td>
<td>Circuit GND potential</td>
</tr>
<tr>
<td>VDD</td>
<td>26</td>
<td>+5V during normal operation.</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td>Main power supply; +5V during operation.</td>
</tr>
<tr>
<td>PROG</td>
<td>25</td>
<td>Output strobe for 8243 I/O expander.</td>
</tr>
<tr>
<td>P10-P17</td>
<td>27-34</td>
<td>8-bit quasi-bidirectional port.</td>
</tr>
<tr>
<td>P20-P23</td>
<td>21-24</td>
<td>8-bit quasi-bidirectional port.</td>
</tr>
<tr>
<td></td>
<td>35-38</td>
<td>P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.</td>
</tr>
<tr>
<td>DB0-DB7</td>
<td>12-19</td>
<td>True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.</td>
</tr>
<tr>
<td>T0</td>
<td>1</td>
<td>Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction</td>
</tr>
<tr>
<td>T1</td>
<td>39</td>
<td>Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction</td>
</tr>
<tr>
<td>INT</td>
<td>6</td>
<td>Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device. Used as a read strobe to external data memory. (Active low)</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>Input which is used to initialize the processor. (Active low) (Non TTL V_H)</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>Output strobe during a bus write. (Active low) Used as write strobe to external data memory.</td>
</tr>
<tr>
<td>ALE</td>
<td>11</td>
<td>Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.</td>
</tr>
<tr>
<td>PSEN</td>
<td>9</td>
<td>Program store enable. This output occurs only during a fetch to external program memory. (Active low)</td>
</tr>
<tr>
<td>SS</td>
<td>5</td>
<td>Single step input can be used in conjunction with ALE to “single step” the processor through each instruction. (Active Low) Used in sync mode</td>
</tr>
<tr>
<td>EA</td>
<td>7</td>
<td>External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high) Used during ROM verification (12V)</td>
</tr>
<tr>
<td>XTAL1</td>
<td>2</td>
<td>One side of crystal input for internal oscillator. Also input for external source. (Non TTL V_H)</td>
</tr>
<tr>
<td>XTAL2</td>
<td>3</td>
<td>Other side of crystal input.</td>
</tr>
</tbody>
</table>
### Accumulator

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A, R</td>
<td>Add register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, @R</td>
<td>Add data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, # data</td>
<td>Add immediate to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A, R</td>
<td>Add register with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, @R</td>
<td>Add data memory with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, # data</td>
<td>Add immediate with carry</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL A, R</td>
<td>And register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, @R</td>
<td>And data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, # data</td>
<td>And immediate to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORL A, R</td>
<td>Or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, @R</td>
<td>Or data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, # data</td>
<td>Or immediate to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>XRL A, R</td>
<td>Exclusive or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, @R</td>
<td>Exclusive or data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, # data</td>
<td>Exclusive or immediate to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Registers

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC R</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC @R</td>
<td>Increment data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC R</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Branch

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP addr</td>
<td>Jump unconditional</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JMPP @A</td>
<td>Jump indirect</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ R, addr</td>
<td>Decrement register and skip</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JC addr</td>
<td>Jump on carry = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC addr</td>
<td>Jump on carry = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JZ addr</td>
<td>Jump on A zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNZ addr</td>
<td>Jump on A not zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTO addr</td>
<td>Jump on T0 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT0 addr</td>
<td>Jump on T0 = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTF addr</td>
<td>Jump on timer flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNi addr</td>
<td>Jump on INT = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JBb addr</td>
<td>Jump on accumulator bit</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### Subroutine

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL addr</td>
<td>Jump to subroutine</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RETR</td>
<td>Return and restore status</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

### Flags

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>Clear carry</td>
<td>.1</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F0</td>
<td>Clear flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td>Complement flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F1</td>
<td>Clear flag 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F1</td>
<td>Complement flag 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Input/Output

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN A, P</td>
<td>Input port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUTL P, A</td>
<td>Output A to port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL P, # data</td>
<td>And immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL P, # data</td>
<td>Or immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INS A, BUS</td>
<td>Input BUS to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUTL BUS, A</td>
<td>Output A to BUS</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL BUS, # data</td>
<td>And immediate to BUS</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL BUS, # data</td>
<td>Or immediate to BUS</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOVDA P, A</td>
<td>Input expander port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVDP A,</td>
<td>Output A to expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANLDP A,</td>
<td>And A to expander port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ORLDP A,</td>
<td>Or A to expander port</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
### Table 2. Instruction Set (Continued)

#### Data Moves

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, R</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, @R</td>
<td>Move data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, # data</td>
<td>Move immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV R, A</td>
<td>Move A to register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV @R, A</td>
<td>Move A to data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV R, # data</td>
<td>Move immediate to register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV @R, # data</td>
<td>Move immediate to data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A, PSW</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV PSW, A</td>
<td>Move A to PSW</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, R</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, @R</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHD A, @R</td>
<td>Exchange nibble of A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVX A, @R</td>
<td>Move external data memory to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVX @R, A</td>
<td>Move A to external data memory</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOV P, @A</td>
<td>Move to A from current page</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOV P3 A, @</td>
<td>Move to A from page 3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Timer/Counter

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, T</td>
<td>Read timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV T, A</td>
<td>Load timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT T</td>
<td>Start timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT CNT</td>
<td>Start timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STOP TCNT</td>
<td>Stop timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN TCNT</td>
<td>Enable timer/counter interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS TCNT</td>
<td>Disable timer/counter interrupt</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Control

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN I</td>
<td>Enable external interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS I</td>
<td>Disable external interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB0</td>
<td>Select register bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB1</td>
<td>Select register bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL MB0</td>
<td>Select memory bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL MB1</td>
<td>Select memory bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ENTO CLK</td>
<td>Enable clock output on T0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Mnemonic

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IDL</td>
<td>Select Idle Operation</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
ABSOLUTE MAXIMUM RATINGS*
Ambient Temperature Under Bias . . . 0°C to 70°C
Storage Temperature ................ -65°C to +150°C
Voltage On Any Pin With Respect to Ground ....................... -0.5V to +7V
Power Dissipation .................... 1.5 Watt

*NORE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS: (TA = 0°C to 70°C; VCC = VDD = 5V ± 10%; VSS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Test Conditions</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage (All Except RESET, X1, X2)</td>
<td>Min: -.5</td>
<td>Typ: .8</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VIL1</td>
<td>Input Low Voltage (RESET, X1, X2)</td>
<td>Min: -.5</td>
<td>Typ: .6</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (All Except XTAL1, XTAL2, RESET)</td>
<td>Min: 2.0</td>
<td>Typ: VCC</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage (X1, X2, RESET)</td>
<td>Min: 3.8</td>
<td>Typ: VCC</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage (BUS)</td>
<td>Min: .45</td>
<td>Typ: V</td>
<td>IOH = 2.0 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage (RD, WR, PSEN, ALE)</td>
<td>Min: .45</td>
<td>Typ: V</td>
<td>IOH = 1.8 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOL2</td>
<td>Output Low Voltage (PROG)</td>
<td>Min: .45</td>
<td>Typ: V</td>
<td>IOH = 1.0 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOL3</td>
<td>Output Low Voltage (All Other Outputs)</td>
<td>Min: .45</td>
<td>Typ: V</td>
<td>IOH = 1.6 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage (BUS)</td>
<td>Min: 2.4</td>
<td>Typ: V</td>
<td>IOH = -400 μA</td>
<td>All</td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage (RD, WR, PSEN, ALE)</td>
<td>Min: 2.4</td>
<td>Typ: V</td>
<td>IOH = -100 μA</td>
<td>All</td>
</tr>
<tr>
<td>VOH2</td>
<td>Output High Voltage (All Other Outputs)</td>
<td>Min: 2.4</td>
<td>Typ: V</td>
<td>IOH = -40 μA</td>
<td>All</td>
</tr>
</tbody>
</table>
D.C. CHARACTERISTICS: \((T_A = 0^\circ C \text{ to } 70^\circ C; \ V_{CC} = V_{DD} = 5V \pm 10\%; \ V_{SS} = 0V)\) (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_{L1})</td>
<td>Leakage Current (T1, INT)</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td>(I_{L1})</td>
<td>Input Leakage Current (P10-P17, P20-P27, EA, SS)</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td>(I_{L2})</td>
<td>Input Leakage Current RESET</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td>(I_{L0})</td>
<td>Leakage Current (BUS, TO) (High Impedance State)</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td>(I_{DD})</td>
<td>V_{DD} Supply Current (RAM Standby)</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>5</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>7</td>
<td>mA</td>
</tr>
<tr>
<td>(I_{DD} + I_{CC})</td>
<td>Total Supply Current*</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30</td>
<td>65</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>35</td>
<td>70</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40</td>
<td>80</td>
<td>mA</td>
</tr>
<tr>
<td>(V_{DD})</td>
<td>RAM Standby Voltage</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
</tr>
</tbody>
</table>

\*\(I_{CC} + I_{DD}\) is measured with all outputs disconnected; SS, RESET, and INT equal to \(V_{CC}\); EA equal to \(V_{SS}\).
A.C. CHARACTERISTICS: \( (T_A = 0^\circ\text{C to } 70^\circ\text{C}; V_{CC} = V_{DD} = 5V \pm 10\%; V_{SS} = 0V) \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>( f (t) ) (Note 3)</th>
<th>11 MHz</th>
<th>Conditions (Note 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>Clock Period</td>
<td>( 1/\text{xal freq} )</td>
<td>Min 90.9 Max 1000 ns</td>
<td>(Note 3)</td>
</tr>
<tr>
<td>( t_{LL} )</td>
<td>ALE Pulse Width</td>
<td>3.5t-170 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{AL} )</td>
<td>Addr Setup to ALE</td>
<td>2t-110 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{A} )</td>
<td>Addr Hold from ALE</td>
<td>( t-40 ) ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{CC1} )</td>
<td>Control Pulse Width (RD, WR)</td>
<td>7.5t-200 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{CC2} )</td>
<td>Control Pulse Width (PSEN)</td>
<td>6t-200 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{DW} )</td>
<td>Data Setup before WR</td>
<td>6.5t-200 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{WD} )</td>
<td>Data Hold after WR</td>
<td>( t-50 ) ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{DR} )</td>
<td>Data Hold (RD, PSEN)</td>
<td>( 1.5t-30 ) ns</td>
<td>Min 0 Max 110 ns</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{RD1} )</td>
<td>RD to Data in</td>
<td>6t-170 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{RD2} )</td>
<td>PSEN to Data in</td>
<td>4.5t-170 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{AW} )</td>
<td>Addr Setup to WR</td>
<td>5t-150 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{AD1} )</td>
<td>Addr Setup to Data (RD)</td>
<td>10.5t-220 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{AD2} )</td>
<td>Addr Setup to Data (PSEN)</td>
<td>7.5t-200 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{AFC1} )</td>
<td>Addr Float to RD, WR</td>
<td>2t-40 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{AFC2} )</td>
<td>Addr Float to PSEN</td>
<td>5.5t-40 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{LAFCD1} )</td>
<td>ALE to Control (RD, WR)</td>
<td>3t-75 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{LAFCD2} )</td>
<td>ALE to Control (PSEN)</td>
<td>1.5t-75 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{CA1} )</td>
<td>Control to ALE (RD, WR, PROG),</td>
<td>( t-65 ) ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{CA2} )</td>
<td>Control to ALE (PSEN)</td>
<td>4t-70 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{CP} )</td>
<td>Port Control Setup to PROG</td>
<td>1.5t-80 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PC} )</td>
<td>Port Control Hold to PROG</td>
<td>4t-260 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PR} )</td>
<td>PROG to P2 Input Valid</td>
<td>8.5t-120 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PF} )</td>
<td>Input Data Hold from PROG</td>
<td>1.5t</td>
<td>Min 0 Max 140 ns</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{DP} )</td>
<td>Output Data Setup</td>
<td>6t-290 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PD} )</td>
<td>Output Data Hold</td>
<td>1.5t-90 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PP} )</td>
<td>PROG Pulse Width</td>
<td>10.5t-250 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PL} )</td>
<td>Port 2 I/O Setup to ALE</td>
<td>4t-200 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{LP} )</td>
<td>Port 2 I/O Hold to ALE</td>
<td>5t-30 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PV} )</td>
<td>Port Output from ALE</td>
<td>4.5t+100 ns</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{OPRR} )</td>
<td>T0 Rep Rate</td>
<td>3t</td>
<td>Min 270 ns</td>
<td>ns</td>
</tr>
<tr>
<td>( t_{CY} )</td>
<td>Cycle Time</td>
<td>15t</td>
<td>Min 1.36 Max 15.0 ( \mu\text{s} )</td>
<td>ns</td>
</tr>
</tbody>
</table>

Notes:
1. Control Outputs CL = 80pF
2. BUS High Impedance Load 20pF
3. \( f(t) \) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.
WAVEFORMS

Instruction Fetch From Program Memory

Read From External Data Memory

Write To External Data Memory

Input And Output For A.C. Tests

PORT 1/PORT 2 TIMING

A C testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".
CRYSTAL OSCILLATOR MODE

C1 = 5pF ± 1/2pF + (STRAY < 5pF)
C2 = (CRYSTAL + STRAY) < 8pF
C3 = 20pF ± 1pF + (STRAY < 5pF)

Crystal series resistance should be less than 30Ω at 11 MHz;
less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

CERAMIC RESONATOR MODE

C1 = C2 = 33pF ± 5%

Driving from external source

For XTAL1 and XTAL2 define “high” as voltages above 1.6V
and “low” as voltages below 1.6V. The duty cycle require­
ments for externally driving XTAL1 and XTAL2 using the
circuit shown above are as follows: XTAL1 must be high 35–
65% of the period and XTAL2 must be high 36–65% of the
period. Rise and fall times must be faster than 20 nS.
SUGGESTED ROM VERIFICATION ALGORITHM FOR H-MOS DEVICE ONLY

INITIAL ROM DUMP CYCLE

-12V
EA

ALE
(NOTE 1)

DB
ADDRESS
ROM DATA
(OUTPUT)
(OUTPUT)

(OUTPUT)

SUBSEQUENT ROM DUMP CYCLES

ADDRESS

RESET

P20-P23

48H 49H 50H
A10 0 ADDR ADDR
A11 0 0 ADDR

VCC = VDD = +5V
VSS = 0V

NOTE: ALE is function of X1, X2 inputs.
8748H/8035H/8749H/8039H
HMOS-E SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- High Performance HMOS-E
- Interval Timer/Event Counter
- Two Single Level Interrupts
- Single 5-Volt Supply
- Over 96 Instructions; 90% Single Byte
- Compatible with 8080/8085
- Easily Expandable Memory and I/O
- Up to 1.35 µSec Instruction Cycle
- All Instructions 1 or 2 cycles

The Intel 8749H/8039H/8748H/8035H are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS-E process.

The family contains 27 I/O lines, an 8-bit timer/counter, on-chip RAM and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

<table>
<thead>
<tr>
<th>Device</th>
<th>Internal Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>8039H</td>
<td>none</td>
</tr>
<tr>
<td>8035H</td>
<td>none</td>
</tr>
<tr>
<td>8749H</td>
<td>2K x 8 EPROM</td>
</tr>
<tr>
<td>8748H</td>
<td>1K x 8 EPROM</td>
</tr>
</tbody>
</table>

![Figure 1. Block Diagram](image1)

![Figure 2. Logic Symbol](image2)

![Figure 3. Pin Configuration](image3)
Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSS</td>
<td>20</td>
<td>Circuit GND potential</td>
<td>All</td>
</tr>
<tr>
<td>VDD</td>
<td>26</td>
<td>+5V during normal operation.</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Programming power supply (+21V).</td>
<td>8748H 8749H</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td>Main power supply; +5V during operation and programming.</td>
<td>All</td>
</tr>
<tr>
<td>PROG</td>
<td>25</td>
<td>Output strobe for 8243 I/O expander.</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Program pulse (+18V) input pin during programming.</td>
<td>8748H 8749H (See Note)</td>
</tr>
<tr>
<td>P10-P17 Port 1</td>
<td>27-34</td>
<td>8-bit quasi-bidirectional port.</td>
<td>All</td>
</tr>
<tr>
<td>P20-P23 P24-P27 Port 2</td>
<td>21-24 35-38</td>
<td>8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.</td>
<td>All</td>
</tr>
<tr>
<td>DB0-DB7 BUS</td>
<td>12-19</td>
<td>True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.</td>
<td>All</td>
</tr>
<tr>
<td>(Cont')</td>
<td></td>
<td>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.</td>
<td></td>
</tr>
<tr>
<td>T0</td>
<td>1</td>
<td>Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction.</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used during programming.</td>
<td>8748H 8749H</td>
</tr>
<tr>
<td>T1</td>
<td>39</td>
<td>Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.</td>
<td>All</td>
</tr>
<tr>
<td>INT</td>
<td>6</td>
<td>Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.</td>
<td>All</td>
</tr>
</tbody>
</table>
# Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD</td>
<td>8</td>
<td>Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device. Used as a read strobe to external data memory. (Active low)</td>
<td>All</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>Input which is used to initialize the processor. (Active low) (Non TTL (V_{IH})) Used during programming.</td>
<td>All</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>Output strobe during a bus write. (Active low) Used as write strobe to external data memory.</td>
<td>All</td>
</tr>
<tr>
<td>ALE</td>
<td>11</td>
<td>Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory</td>
<td>All</td>
</tr>
<tr>
<td>PSEN</td>
<td>9</td>
<td>Program store enable. This output occurs only during a fetch to external program memory. (Active low)</td>
<td>ALL</td>
</tr>
<tr>
<td>SS</td>
<td>5</td>
<td>Single step input. Can be used in conjunction with ALE to “single step” the processor through each instruction.</td>
<td>All</td>
</tr>
<tr>
<td>EA</td>
<td>7</td>
<td>External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high) Used during (18V) programming</td>
<td>All</td>
</tr>
<tr>
<td>XTAL1</td>
<td>2</td>
<td>One side of crystal input for internal oscillator. Also input for external source. (Non TTL (V_{IH}))</td>
<td>All</td>
</tr>
<tr>
<td>XTAL2</td>
<td>3</td>
<td>Other side of crystal input</td>
<td>All</td>
</tr>
</tbody>
</table>

**NOTE:** On the 8749H/8039H, PROG must be clamped to \(V_{CC}\) when not programming. A diode should be used when using an 8243; otherwise, a direct connection is permissible.
**Table 2. Instruction Set**

### Accumulator

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A, R</td>
<td>Add register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, @R</td>
<td>Add data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, # data</td>
<td>Add immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADDC A, R</td>
<td>Add register with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, @R</td>
<td>Add memory with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, # data</td>
<td>Add immediate with carry</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL A, R</td>
<td>And register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, @R</td>
<td>And data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, # data</td>
<td>And immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL A, @R</td>
<td>Or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A # data</td>
<td>Or data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, # data</td>
<td>Or immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XRL A, R</td>
<td>Exclusive or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, @R</td>
<td>Exclusive or data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL, A, # data</td>
<td>Exclusive or immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Registers

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC R</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC @R</td>
<td>Increment data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC R</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Branch

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP addr</td>
<td>Jump unconditional</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JMPP @A</td>
<td>Jump indirect</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ R, addr</td>
<td>Decrement register and skip</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JR addr</td>
<td>Jump on carry = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC addr</td>
<td>Jump on carry = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JZ addr</td>
<td>Jump on A zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNZ addr</td>
<td>Jump on A not zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JT0 addr</td>
<td>Jump on T0 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT0 addr</td>
<td>Jump on T0 = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JT1 addr</td>
<td>Jump on T1 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT1 addr</td>
<td>Jump on T1 = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF0 addr</td>
<td>Jump on F0 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF1 addr</td>
<td>Jump on F1 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTF addr</td>
<td>Jump on timer flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JN1 addr</td>
<td>Jump on INT = 0</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### Subroutine

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL addr</td>
<td>Jump to subroutine</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RETR</td>
<td>Return and restore status</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

### Flags

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>Clear carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F0</td>
<td>Clear flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td>Complement flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F1</td>
<td>Clear flag 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F1</td>
<td>Complement flag 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 2. Instruction Set (Continued)

#### Data Moves

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, R</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, @R</td>
<td>Move data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, #data</td>
<td>Move immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV R, A</td>
<td>Move A to register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV @R, A</td>
<td>Move A to data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV R, #data</td>
<td>Move immediate to register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV @R, #data</td>
<td>Move immediate to data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A, PSW</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV PSW, A</td>
<td>Move A to PSW</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, R</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, @R</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHD A, @R</td>
<td>Exchange nibble of A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVX A, @R</td>
<td>Move external data memory to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVX @R, A</td>
<td>Move A to external data memory</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVPA, @A</td>
<td>Move to A from current page</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVPA3 A, @A</td>
<td>Move to A from page 3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Timer/Counter

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, T</td>
<td>Read timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV T, A</td>
<td>Load timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT T</td>
<td>Start timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT CNT</td>
<td>Start counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STOP TCNT</td>
<td>Stop timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN TCNTI</td>
<td>Enable timer/counter interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS TCNTI</td>
<td>Disable timer/counter interrupt</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Control

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN I</td>
<td>Enable external interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS I</td>
<td>Disable external interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB0</td>
<td>Select register bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB1</td>
<td>Select register bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL MB0</td>
<td>Select memory bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL MB1</td>
<td>Select memory bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ENTO CLK</td>
<td>Enable clock output on T0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
ABSOLUTE MAXIMUM RATINGS*
Ambient Temperature Under Bias . . . 0° C to 70° C
Storage Temperature ............. -65° C to +150° C
Voltage On Any Pin With Respect
to Ground .................. -0.5V to +7V
Power Dissipation .................. 1.0 Watt

"NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS: (T_A = 0° C to 70° C; V_CC = V_DD = 5V ± 10%; V_SS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Test Conditions</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
<td></td>
</tr>
<tr>
<td>VIL</td>
<td>Input Low Voltage (All Except RESET, X1, X2)</td>
<td>-.5</td>
<td>.8</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VIL1</td>
<td>Input Low Voltage (RESET, X1, X2)</td>
<td>-.5</td>
<td>.6</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (All Except XTAL1, XTAL2, RESET)</td>
<td>2.0</td>
<td>V_CC</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage (X1, X2, RESET)</td>
<td>3.8</td>
<td>V_CC</td>
<td>V</td>
<td>All</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage (BUS)</td>
<td>.45</td>
<td>V</td>
<td>I_OH = 2.0 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage (RD, WR, PSEN, ALE)</td>
<td>.45</td>
<td>V</td>
<td>I_OH = 1.8 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOL2</td>
<td>Output Low Voltage (PROG)</td>
<td>.45</td>
<td>V</td>
<td>I_OH = 1.0 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOL3</td>
<td>Output Low Voltage (All Other Outputs)</td>
<td>.45</td>
<td>V</td>
<td>I_OH = 1.6 mA</td>
<td>All</td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage (BUS)</td>
<td>2.4</td>
<td>V</td>
<td>I_OH = -400 µA</td>
<td>All</td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage (RD, WR, PSEN, ALE)</td>
<td>2.4</td>
<td>V</td>
<td>I_OH = -100 µA</td>
<td>All</td>
</tr>
<tr>
<td>VOH2</td>
<td>Output High Voltage (All Other Outputs)</td>
<td>2.4</td>
<td>V</td>
<td>I_OH = -40 µA</td>
<td>All</td>
</tr>
</tbody>
</table>
**D.C. CHARACTERISTICS:** (TA = 0°C to 70°C; VCC = VDD = 5V ± 10%; VSS = 0V) (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Unit</th>
<th>Test Conditions</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>IL1</td>
<td>Leakage Current (T1, INT)</td>
<td>±10 μA</td>
<td>VSS ≤ VIN ≤ VCC</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>IL11</td>
<td>Input Leakage Current (P10–P17, P20–P27, EA, SS)</td>
<td>-500 μA</td>
<td>VSS + .45 ≤ VIN ≤ VCC</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>IL2</td>
<td>Input Leakage Current RESET</td>
<td>-10 μA</td>
<td>VSS ≤ VIN ≤ 3.8V</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>IL0</td>
<td>Leakage Current (BUS, T0) (High Impedance State)</td>
<td>±10 μA</td>
<td>VSS ≤ VIN ≤ VCC</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>IDD + ICC</td>
<td>Total Supply Current*</td>
<td>80 100 mA</td>
<td>8035H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>95 110 mA</td>
<td>8039H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>80 100 mA</td>
<td>8748H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>95 110 mA</td>
<td>8749H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*ICC + IDD is measured with all outputs disconnected; SS, RESET, and INT equal to VCC; EA equal to VSS.
### A.C. CHARACTERISTICS:

\( TA = 0°C \) to \( 70°C \); \( V_{CC} = V_{DD} = 5V \pm 10\%; \ V_{SS} = 0V \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>( f(t) ) (Note 3)</th>
<th>( 11 \text{ MHz} )</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Conditions (Note 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>Clock Period</td>
<td>( 1/\text{xtal freq} )</td>
<td>90.9</td>
<td>1000</td>
<td>ns</td>
<td>(Note 3)</td>
<td></td>
</tr>
<tr>
<td>( t_{LL} )</td>
<td>ALE Pulse Width</td>
<td>3.5t–170</td>
<td>150</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AL} )</td>
<td>Addr Setup to ALE</td>
<td>2t–110</td>
<td>70</td>
<td>ns</td>
<td></td>
<td>(Note 2)</td>
<td></td>
</tr>
<tr>
<td>( t_{LA} )</td>
<td>Addr Hold from ALE</td>
<td>( t–40 )</td>
<td>50</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CC1} )</td>
<td>Control Pulse Width (RD, WR)</td>
<td>7.5t–200</td>
<td>480</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CC2} )</td>
<td>Control Pulse Width (PSEN)</td>
<td>6t–200</td>
<td>350</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DW} )</td>
<td>Data Setup before WR</td>
<td>6.5t–200</td>
<td>390</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{WD} )</td>
<td>Data Hold after WR</td>
<td>( t–50 )</td>
<td>40</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DR} )</td>
<td>Data Hold (RD, PSEN)</td>
<td>1.5t–30</td>
<td>0</td>
<td>110</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RD1} )</td>
<td>RD to Data in</td>
<td>6t–170</td>
<td>375</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{RD2} )</td>
<td>PSEN to Data in</td>
<td>4.5t–170</td>
<td>240</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AW} )</td>
<td>Addr Setup to WR</td>
<td>5t–150</td>
<td>300</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AD1} )</td>
<td>Addr Setup to Data (RD)</td>
<td>10.5t–220</td>
<td>730</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AD2} )</td>
<td>Addr Setup to Data (PSEN)</td>
<td>7.5t–200</td>
<td>460</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{AFC1} )</td>
<td>Addr Float to RD, WR</td>
<td>2t–40</td>
<td>140</td>
<td>ns</td>
<td></td>
<td>(Note 2)</td>
<td></td>
</tr>
<tr>
<td>( t_{AFC2} )</td>
<td>Addr Float to PSEN</td>
<td>( .5t–40 )</td>
<td>0</td>
<td>10</td>
<td>ns</td>
<td>(Note 2)</td>
<td></td>
</tr>
<tr>
<td>( t_{LAFC1} )</td>
<td>ALE to Control (RD, WR)</td>
<td>3t–75</td>
<td>200</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{LAFC2} )</td>
<td>ALE to Control (PSEN)</td>
<td>1.5t–75</td>
<td>60</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CA1} )</td>
<td>Control to ALE (RD, WR, PROG)</td>
<td>( t–65 )</td>
<td>25</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CA2} )</td>
<td>Control to ALE (PSEN)</td>
<td>4t–70</td>
<td>290</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CP} )</td>
<td>Port Control Setup to PROG</td>
<td>1.5t–80</td>
<td>50</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PC} )</td>
<td>Port Control Hold to PROG</td>
<td>4t–260</td>
<td>100</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PR} )</td>
<td>PROG to P2 Input Valid</td>
<td>8.5t–120</td>
<td>650</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PF} )</td>
<td>Input Data Hold from PROG</td>
<td>1.5t</td>
<td>0</td>
<td>140</td>
<td>ns</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{DP} )</td>
<td>Output Data Setup</td>
<td>6t–290</td>
<td>250</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PD} )</td>
<td>Output Data Hold</td>
<td>1.5t–90</td>
<td>40</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PP} )</td>
<td>PROG Pulse Width</td>
<td>10.5t–250</td>
<td>700</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PL} )</td>
<td>Port 2 I/O Setup to ALE</td>
<td>4t–200</td>
<td>160</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{LP} )</td>
<td>Port 2 I/O Hold to ALE</td>
<td>( .5t–30 )</td>
<td>15</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{PV} )</td>
<td>Port Output from ALE</td>
<td>4.5t+100</td>
<td>510</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{OPRR} )</td>
<td>T0 Rep Rate</td>
<td>3t</td>
<td>270</td>
<td>ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_{CY} )</td>
<td>Cycle Time</td>
<td>15t</td>
<td>1.36</td>
<td>15.0</td>
<td>( \mu s )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**

1. Control Outputs CL = 80pF  
2. BUS High Impedance Load 20pF  
3. \( f(t) \) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.
WAVEFORMS

Instruction Fetch From Program Memory

Read From External Data Memory

Input And Output For A.C. Tests

A.C. testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

PORT 1/PORT 2 TIMING

1ST CYCLE

2ND CYCLE

1LP

1PL

1PV

1LA

1LL

1LAF2

1LAC1

1LFA1

1LD2

1RD2

1RD1

1LAFC2

1CC2

1CA2

1ICC1

1CA1

2.4V

0.45V

TEST POINTS

0.8V
CRYSTAL OSCILLATOR MODE

\[ C_1 = 5\text{pF} \pm 1/2\text{pF} + (\text{STRAY} < 5\text{pF}) \]
\[ C_2 = (\text{CRYSTAL} + \text{STRAY}) < 8\text{pF} \]
\[ C_3 = 20\text{pF} \pm 1\text{pF} + (\text{STRAY} < 5\text{pF}) \]

Crystal series resistance should be less than 30Ω at 11 MHz;
less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

CERAMIC RESONATOR MODE

\[ C_1 = C_2 = 33\text{pF} \pm 5\% \]
\[ C_3 = 20\text{pF} \pm 1\text{pF} + (\text{STRAY} < 5\text{pF}) \]

DRIVING FROM EXTERNAL SOURCE

For XTAL1 and XTAL2 define "high" as voltages above 1.6V
and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the
circuit shown above are as follows: XTAL1 must be high 35–65% of the period and XTAL2 must be high 36–65% of the
period. Rise and fall times must be faster than 20 nS.

PROGRAMMING, VERIFYING, AND ERASING THE 8749H (8748H) EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL 1</td>
<td>Clock Input (3 to 4.0 MHz)</td>
</tr>
<tr>
<td>XTAL 2</td>
<td></td>
</tr>
<tr>
<td>Reset</td>
<td>Initialization and Address Latching</td>
</tr>
<tr>
<td>Test 0</td>
<td>Selection of Program or Verify Mode</td>
</tr>
<tr>
<td>EA</td>
<td>Activation of Program/Verify Modes</td>
</tr>
<tr>
<td>BUS</td>
<td>Address and Data Input</td>
</tr>
<tr>
<td>P20-P22</td>
<td>Data Output During Verify</td>
</tr>
<tr>
<td>VDD</td>
<td>Address Input</td>
</tr>
<tr>
<td>PROG</td>
<td>Programming Power Supply</td>
</tr>
<tr>
<td></td>
<td>Program Pulse Input</td>
</tr>
</tbody>
</table>

WARNING:
An attempt to program a missocketed 8749H (8748H) will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. \( V_{DD} = 5V \) - Clock applied or internal oscillator operating; \( \text{RESET} = 0V \), \( \text{TEST} 0 = 5V \), \( \text{EA} = 5V \), \( \text{BUS} \) and \( \text{PROG} \) floating. \( \text{P10} \) and \( \text{P11} \) must be tied to ground.
2. Insert 8749H (8748H) in programming socket.
3. \( \text{TEST} 0 = 0V \) (select program mode)
4. \( \text{EA} = 18V \) (activate program mode)
5. Address applied to BUS and P20–22
6. \( \text{RESET} = 5V \) (latch address)
7. Data applied to BUS
8. \( V_{DD} = 21V \) (programming power)
9. \( \text{PROG} = V_{CC} \) or float followed by one 50ms pulse to 18V
10. \( V_{DD} = 5V \)
11. \( \text{TEST} 0 = 5V \) (verify mode)
12. Read and verify data on BUS
13. \( \text{TEST} 0 = 0V \)
14. \( \text{RESET} = 0V \) and repeat from step 5
15. Programmer should be at conditions of step 1 when 8749H (8748H) is removed from socket.
### A.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:

(TA = 25°C ± 5°C; VCC = 5V ± 5%; VDD = 21 ± .5V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tAW</td>
<td>Address Setup Time to RESET†</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWA</td>
<td>Address Hold Time After RESET†</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tDW</td>
<td>Data in Setup Time to PROG1</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWD</td>
<td>Data in Hold Time After PROG1</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tPH</td>
<td>RESET Hold Time to Verify</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tVDDW</td>
<td>VDD Hold Time Before PROG1</td>
<td>0</td>
<td>1.0</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>tVDDH</td>
<td>VDD Hold Time After PROG1</td>
<td>0</td>
<td>1.0</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>tpW</td>
<td>Program Pulse Width</td>
<td>50</td>
<td>60</td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>tTW</td>
<td>Test 0 Setup Time for Program Mode</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWT</td>
<td>Test 0 Hold Time After Program Mode</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tDO</td>
<td>Test 0 to Data Out Delay</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tWW</td>
<td>RESET Pulse Width to Latch Address</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t,R,tF</td>
<td>VDD and PROG Rise and Fall Times</td>
<td>0.5</td>
<td>100</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>tCY</td>
<td>CPU Operation Cycle Time</td>
<td>3.75</td>
<td>5</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>tRE</td>
<td>RESET Setup Time before EA†</td>
<td>4tCY</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** If Test 0 is high, tDO can be triggered by RESET†

### D.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:

(TA = 25°C ± 5°C; VCC = 5V ± 5%; VDD = 21 ± .5V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDDH</td>
<td>VDD Program Voltage High Level</td>
<td>20.5</td>
<td>21.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VDDL</td>
<td>VDD Voltage Low Level</td>
<td>4.75</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VPH</td>
<td>PROG Program Voltage High Level</td>
<td>17.5</td>
<td>18.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VPL</td>
<td>PROG Voltage Low Level</td>
<td>4.0</td>
<td>VCC</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VEAH</td>
<td>EA Program or Verify Voltage High Level</td>
<td>17.5</td>
<td>18.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>IDD</td>
<td>VDD High Voltage Supply Current</td>
<td>20.0</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>IROG</td>
<td>PROG High Voltage Supply Current</td>
<td>1.0</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>IEA</td>
<td>EA High Voltage Supply Current</td>
<td>1.0</td>
<td></td>
<td>mA</td>
<td></td>
</tr>
</tbody>
</table>
WAVEFORMS

COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)

VERIFY MODE
SUGGESTED EPROM VERIFICATION ALGORITHM FOR HMOS-E DEVICE ONLY

INITIAL EPROM DUMP CYCLE

<table>
<thead>
<tr>
<th>ALE (NOTE 1)</th>
<th>12V</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td></td>
</tr>
</tbody>
</table>

SUBSEQUENT EPROM DUMP CYCLES

<table>
<thead>
<tr>
<th>ADDRESS (INPUT)</th>
<th>ROM DATA (OUTPUT)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ADDRESS (INPUT)</th>
<th>ROM DATA (OUTPUT)</th>
</tr>
</thead>
</table>

VCC = VDD = +5V
VSS = 0V

<table>
<thead>
<tr>
<th>48H</th>
<th>49H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A10</td>
<td>ADDR</td>
</tr>
<tr>
<td>A11</td>
<td>0</td>
</tr>
</tbody>
</table>

NOTE: ALE is function of X1, X2 inputs.
The new Intel EXPRESS family of single-component 8-bit microcomputers offers enhanced processing options to the familiar 8048AH/8035AHL, 8748H, 8049AH/8039AHL, 8749H, 8050AH/8040AHL Intel components. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and -40°C to 85°C operation with or without 168 ± 8 hours of dynamic burn-in at 125°C per MIL-STD-883, method 1015. Figure 1 summarizes the option marking designators and package selections.

For a complete description of 8048AH/8035AHL, 8748H, 8049AH/8039AHL, 8749H, 8040AHL and 8050AH features and operating characteristics, refer to the respective standard commercial grade data sheet. This document highlights only the electrical specifications which differ from the respective commercial part.

<table>
<thead>
<tr>
<th>Temp Range °C</th>
<th>0-70</th>
<th>-40-85</th>
<th>0-70</th>
<th>-40-85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burn In</td>
<td>0 Hrs</td>
<td>0 Hrs</td>
<td>168 Hrs</td>
<td>168 Hrs</td>
</tr>
<tr>
<td>P8048AH</td>
<td>TP8048AH</td>
<td></td>
<td>QP8048AH</td>
<td></td>
</tr>
<tr>
<td>D8048AH</td>
<td>TD8048AH</td>
<td></td>
<td>QD8048AH</td>
<td></td>
</tr>
<tr>
<td>D8748H</td>
<td>TD8748H</td>
<td></td>
<td>QD8748H</td>
<td></td>
</tr>
<tr>
<td>P8035AHL</td>
<td>TP8035AHL</td>
<td></td>
<td>QP8035AHL</td>
<td></td>
</tr>
<tr>
<td>D8035AHL</td>
<td>TD8035AHL</td>
<td></td>
<td>QD8035AHL</td>
<td></td>
</tr>
<tr>
<td>P8049AH</td>
<td>TP8049AH</td>
<td></td>
<td>QP8049AH</td>
<td></td>
</tr>
<tr>
<td>D8049AH</td>
<td>TD8049AH</td>
<td></td>
<td>QD8049AH</td>
<td></td>
</tr>
<tr>
<td>D8749H</td>
<td>TD8749H</td>
<td></td>
<td>QD8749H</td>
<td></td>
</tr>
<tr>
<td>P8039AHL</td>
<td>TP8039AHL</td>
<td></td>
<td>QP8039AHL</td>
<td></td>
</tr>
<tr>
<td>D8039AHL</td>
<td>TD8039AHL</td>
<td></td>
<td>QD8039AHL</td>
<td></td>
</tr>
<tr>
<td>P8050AH</td>
<td>TP8050AH</td>
<td></td>
<td>QP8050AH</td>
<td></td>
</tr>
<tr>
<td>D8050AH</td>
<td>TD8050AH</td>
<td></td>
<td>QD8050AH</td>
<td></td>
</tr>
<tr>
<td>P8040AHL</td>
<td>TP8040AHL</td>
<td></td>
<td>QP8040AHL</td>
<td></td>
</tr>
<tr>
<td>D8040AHL</td>
<td>TD8040AHL</td>
<td></td>
<td>QD8040AHL</td>
<td></td>
</tr>
<tr>
<td>P8243</td>
<td>TP8243</td>
<td></td>
<td>QP8243</td>
<td></td>
</tr>
<tr>
<td>D8243</td>
<td>TD8243</td>
<td></td>
<td>QD8243</td>
<td></td>
</tr>
</tbody>
</table>

* Commercial Grade
P Plastic Package
D Cerdip Package
### Extended Temperature Electrical Specification Deviations

TP8048AH/TP8035AH/LP8048AH/LP8035AHL  
TD8048AH/TD8035AHL/LD8048AH/LD8035AHL

**D.C. CHARACTERISTICS:** (\(TA = -40^\circ C\) to \(85^\circ C\); \(V_{CC} = V_{DD} = 5V \pm 10\%; V_{SS} = 0V\))

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th></th>
<th></th>
<th></th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{IH})</td>
<td>Input High Voltage (All Except XTAL1, XTAL2, RESET)</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
<td>Unit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.2</td>
<td>(V_{CC})</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>(I_{DD})</td>
<td>(V_{DD}) Supply Current</td>
<td>4</td>
<td>8</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(I_{DD} + I_{CC})</td>
<td>Total Supply Current</td>
<td>40</td>
<td>80</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

TP8049AH/TP8039AH/LP8049AH/LP8039AHL  
TD8049AH/TD8039AHL/LD8049AH/LD8039AHL

**D.C. CHARACTERISTICS:** (\(TA = -40^\circ C\) to \(85^\circ C\); \(V_{CC} = V_{DD} = 5V \pm 10\%; V_{SS} = 0V\))

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th></th>
<th></th>
<th></th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{IH})</td>
<td>Input High Voltage (All Except XTAL1, XTAL2, RESET)</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
<td>Unit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.2</td>
<td>(V_{CC})</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>(I_{DD})</td>
<td>(V_{DD}) Supply Current</td>
<td>5</td>
<td>10</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(I_{DD} + I_{CC})</td>
<td>Total Supply Current</td>
<td>50</td>
<td>100</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

TP8050AH/TP8040AHL/LP8050AHL/LP8040AHL  
TD8050AH/TD8040AHL/LD8050AHL/LD8040AHL

**D.C. CHARACTERISTICS:** (\(TA = -40^\circ C\) to \(85^\circ C\); \(V_{CC} = V_{DD} = 5V \pm 10\%; V_{SS} = 0V\))

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th></th>
<th></th>
<th></th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{IH})</td>
<td>Input High Voltage (All Except XTAL1, XTAL2, RESET)</td>
<td>Min</td>
<td>Typ</td>
<td>Max</td>
<td>Unit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.2</td>
<td>(V_{CC})</td>
<td></td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>(I_{DD})</td>
<td>(V_{DD}) Supply Current</td>
<td>10</td>
<td>20</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(I_{DD} + I_{CC})</td>
<td>Total Supply Current</td>
<td>75</td>
<td>120</td>
<td>mA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Extended Temperature Electrical Specification Deviations

TD8748H/LD8748H

D.C. CHARACTERISTICS: \( T_A = -40^\circ C \) to \( 85^\circ C \); \( V_{CC} = V_{DD} = 5V \pm 10\%; V_{SS} = 0V \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{IH} )</td>
<td>Input High Voltage (All Except XTAL1, XTAL2, RESET)</td>
<td>Min 2.2 Typ ( V_{CC} ) Max</td>
<td></td>
</tr>
<tr>
<td>( I_{DD} + I_{CC} )</td>
<td>Total Supply Current</td>
<td>Min 50 Typ 130 Max mA</td>
<td></td>
</tr>
</tbody>
</table>

TD8749H/LD8749H

D.C. CHARACTERISTICS: \( T_A = -40^\circ C \) to \( 85^\circ C \); \( V_{CC} = V_{DD} = 5V \pm 10\%; V_{SS} = 0V \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_{IH} )</td>
<td>Input High Voltage (All Except XTAL1, XTAL2, RESET)</td>
<td>Min 2.2 Typ ( V_{CC} ) Max</td>
<td></td>
</tr>
<tr>
<td>( I_{DD} + I_{CC} )</td>
<td>Total Supply Current</td>
<td>Min 75 Typ 150 Max mA</td>
<td></td>
</tr>
</tbody>
</table>

TP8243/TD8243/LD8243

D.C. CHARACTERISTICS: \( T_A = -40^\circ C \) to \( 85^\circ C \); \( V_{CC} = 5V \pm 10\%; V_{SS} = 0V \)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{CC} )</td>
<td>( V_{CC} ) Supply Current</td>
<td>Min 15 Typ 25 Max mA</td>
<td></td>
</tr>
</tbody>
</table>
Extended Temperature Electrical Specification Deviations

TD8022/LD8022

D.C. CHARACTERISTICS: \((T_A = -40^\circ C \text{ to } 85^\circ C; \ V_{CC} = 5.5V \pm 1V; \ V_{SS} = 0V)\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Typ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>V_{TH} - 0.2 V</td>
</tr>
<tr>
<td>V_{IL1}</td>
<td>Input Low Voltage (Port 0)</td>
<td>-0.5</td>
<td>V_{TH} - 0.2 V</td>
</tr>
<tr>
<td>V_{IH}</td>
<td>Input High Voltage (All Except XTAL1, RESET)</td>
<td>2.3 VCC</td>
<td>V</td>
</tr>
<tr>
<td>V_{IH1}</td>
<td>Input High Voltage (All Except XTAL1, RESET)</td>
<td>3.8 VCC</td>
<td>V</td>
</tr>
<tr>
<td>V_{IH2}</td>
<td>Input High Voltage (Port 0)</td>
<td>V_{TH} + 0.2 V</td>
<td>VCC</td>
</tr>
<tr>
<td>V_{IH3}</td>
<td>Input High Voltage (RESET, XTAl1)</td>
<td>3.8 VCC</td>
<td>V</td>
</tr>
<tr>
<td>V_{IL}</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.5 V</td>
</tr>
<tr>
<td>V_{OL}</td>
<td>Output Low Voltage</td>
<td>0.45 V</td>
<td>I_{OL} = 0.8 mA</td>
</tr>
<tr>
<td>V_{OL1}</td>
<td>Output Low Voltage (P10, P11)</td>
<td>2.5 V</td>
<td>I_{OL} = 3 mA</td>
</tr>
<tr>
<td>V_{OH}</td>
<td>Output High Voltage (All unless open drain option Port 0)</td>
<td>2.4 V</td>
<td>I_{OH} = 30 µA</td>
</tr>
<tr>
<td>I_{L1}</td>
<td>Input Current (T1)</td>
<td>±700 µA</td>
<td>V_{CC} = V_{IN} = V_{SS} + 0.45V</td>
</tr>
<tr>
<td>I_{L11}</td>
<td>Input Current to Ports</td>
<td>500 µA</td>
<td>V_{IN} = 0.45V</td>
</tr>
<tr>
<td>I_{CC}</td>
<td>V_{CC} Supply Current</td>
<td>120 mA</td>
<td></td>
</tr>
</tbody>
</table>

A.C. CHARACTERISTICS: \((T_A = -40^\circ C \text{ to } 85^\circ C; \ V_{CC} = 5.5V \pm 1V; \ V_{SS} = 0V)\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>µs</td>
<td>3.58 MHz XTAL = 8.38 µs t_{CY}</td>
</tr>
<tr>
<td>I_{CY}</td>
<td>Cycle Time</td>
<td>8.38</td>
<td>50.0</td>
<td>µs</td>
<td>3.58 MHz XTAL = 8.38 µs t_{CY}</td>
</tr>
<tr>
<td>VT1</td>
<td>Zero-Cross Detection Input (T1)</td>
<td>1</td>
<td>3</td>
<td>VACpp AC Coupled</td>
<td></td>
</tr>
<tr>
<td>AZC</td>
<td>Zero-Cross Accuracy</td>
<td>±200 mV</td>
<td>60 Hz Sine Wave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT1</td>
<td>Zero-Cross Detection Input Frequency (T1)</td>
<td>0.05</td>
<td>1 kHz</td>
<td>60 Hz Sine Wave</td>
<td></td>
</tr>
<tr>
<td>t_{LL}</td>
<td>ALE Pulse Width</td>
<td>3.9</td>
<td>23.0</td>
<td>µs</td>
<td>t_{CY} = 8.38 µs for min</td>
</tr>
</tbody>
</table>

NOTE: Control Outputs: C_L = 80 pF; T_{CY} = 8.38 µsec.

A/D CONVERTER CHARACTERISTICS: \((AV_{CC} = 5.5V \pm 1V; \ AV_{SS} = 0V; \ AV_{CC}/2 \leq V_{AREF} \leq AV_{CC})\)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute Accuracy</td>
<td>Min 1.6% FSR ± ½ LSB</td>
<td>LSB</td>
</tr>
</tbody>
</table>

NOTE: The analog input must be maintained at a constant voltage during the sample time (tss + tsh).
*Refer to individual commercial grade data sheets for complete operating characteristics.
80C49-7/80C39-7
CHMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 80C49-7 Low Power Mask Programmable ROM
- 80C39-7 Low Power, CPU only

- Pin-to-pin Compatible with Intel's 8049AH/8039AHL
- 1.36 μsec Instruction Cycle. All Instructions 1 or 2 Cycles
- Ability to Maintain Operation during AC Power Line Interruptions
- Exit Idle Mode with an External or Internal Interrupt Signal
- Battery Operation
- 3 Power Consumption Selections
  - Normal Operation: 12 mA @ 11 MHz @ 5V
  - Idle Mode: 5 mA @ 11 MHz @ 5V
  - Power Down: 2 μA @ 2.0V
- 11 MHz, TTL Compatible Operation:
  - VCC = 5V ± 10%
  - CMOS Compatible Operation;
    - VCC = 5V ± 20%

Intel's 80C49-7/80C39-7 are low power, CHMOS versions of the popular MCS®-48 HMOS family members. CHMOS is a technology built on HMOS II and features high resistivity P substrate, diffused N well, and scaled N and P channel devices.

The 80C49-7/80C39-7 have been designed to provide low power consumption and high performance.

The 80C49-7 contains a 2K x 8 program memory, a 128 x 8 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to an on-board oscillator and clock circuits. For systems that require extra capability, the 80C49-7 can be expanded using CMOS external memories and MCS®-80 and MCS®-85 peripherals. The 80C39-7 is the equivalent of the 80C49-7 without program memory on-board.

The CHMOS design of the 80C49-7 opens new application areas that require battery operation, low power standby, wide voltage range, and the ability to maintain operation during AC power line interruptions. These applications include portable and hand-held instruments, telecommunications, consumer, and automotive.

---

**Figure 1.** Block Diagram

**Figure 2.** Logic Symbol

**Figure 3.** Pin Configuration
### Table 1. Pin Description

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSS</td>
<td>20</td>
<td>Circuit GND potential</td>
</tr>
<tr>
<td>VDD</td>
<td>26</td>
<td>Low Power standby pin</td>
</tr>
<tr>
<td>VCC</td>
<td>40</td>
<td>Main power supply; +5V during operation</td>
</tr>
<tr>
<td>PROG</td>
<td>25</td>
<td>Output strobe for 82C43 I/O expander</td>
</tr>
<tr>
<td>P10-P17 Port 1</td>
<td>27-34</td>
<td>8-bit quasi-bidirectional port</td>
</tr>
<tr>
<td>P20-P23</td>
<td>21-24</td>
<td>8-bit quasi-bidirectional port</td>
</tr>
<tr>
<td>P24-P27 Port 2</td>
<td>35-38</td>
<td>P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.</td>
</tr>
<tr>
<td>DB0-DB7 BUS</td>
<td>12-19</td>
<td>True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.</td>
</tr>
<tr>
<td>T0</td>
<td>1</td>
<td>Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction</td>
</tr>
<tr>
<td>T1</td>
<td>39</td>
<td>Input pin testable using the JT1, and JNT1 instructions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>6</td>
<td>Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) Interrupt must remain low for at least 3 machine cycles for proper operation.</td>
</tr>
<tr>
<td>RD</td>
<td>8</td>
<td>Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device. Used as a read strobe to external data memory. (Active low)</td>
</tr>
<tr>
<td>RESET</td>
<td>4</td>
<td>Input which is used to initialize the processor. (Active low) (Non TTL V\text{IH})</td>
</tr>
<tr>
<td>WR</td>
<td>10</td>
<td>Output strobe during a bus write. (Active low) Used as write strobe to external data memory.</td>
</tr>
<tr>
<td>ALE</td>
<td>11</td>
<td>Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.</td>
</tr>
<tr>
<td>PSEN</td>
<td>9</td>
<td>Program store enable. This output occurs only during a fetch to external program memory. (Active low)</td>
</tr>
<tr>
<td>SS</td>
<td>5</td>
<td>Single step input can be used in conjunction with</td>
</tr>
</tbody>
</table>
Table 1. Pin Description (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS (Con’t)</td>
<td></td>
<td>ALE to “single step” the processor through each instruction (Active low)</td>
</tr>
<tr>
<td>EA</td>
<td>7</td>
<td>External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTAL1</td>
<td>2</td>
<td>One side of crystal input for internal oscillator. Also input for external source. (Non TTL VIH)</td>
</tr>
<tr>
<td>XTAL2</td>
<td>3</td>
<td>Other side of crystal input</td>
</tr>
</tbody>
</table>

**IDLE MODE DESCRIPTION**

The 80C49-7, when placed into Idle mode, keeps the oscillator, the internal timer and the external interrupt and counter pins functioning and maintains the internal register and RAM status.

To place the 80C49-7 in Idle mode, a command instruction (op code 01H) is executed. To terminate Idle mode, a reset must be performed or interrupts must be enabled and an interrupt signal generated. There are two interrupt sources that can restore normal operation. One is an external signal applied to the interrupt pin. The other is from the overflow of the timer/counter. When either interrupt is invoked, the CPU is taken out of Idle mode and vectors to the interrupt’s service routine address. Along with the Idle mode, the standard MCS®-48 power-down mode is still maintained.
# Table 2. Instruction Set

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A, R</td>
<td>Add register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, @R</td>
<td>Add data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, # data</td>
<td>Add immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ADD A, R</td>
<td>Add register with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, @R</td>
<td>Add data memory with carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, # data</td>
<td>Add immediate with carry</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL A, R</td>
<td>And register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, @R</td>
<td>And data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, R</td>
<td>Or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, # data</td>
<td>Or immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL A, @R</td>
<td>Or data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, R</td>
<td>Exclusive or register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, @R</td>
<td>Exclusive or data memory to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR A</td>
<td>Clear A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL A</td>
<td>Complement A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DA A</td>
<td>Decimal adjust A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWAP A</td>
<td>Swap nibbles of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate A left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate A left through carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RR A</td>
<td>Rotate A right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RRC A</td>
<td>Rotate A right through carry</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVDA, P</td>
<td>Input expander port to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVDP, A</td>
<td>Output A to port</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL P, # data</td>
<td>And immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL P, # data</td>
<td>Or immediate to port</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>INS A, BUS</td>
<td>Input BUS to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUTL BUS, A</td>
<td>Output A to BUS</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ANL BUS, # data</td>
<td>And immediate to BUS</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL BUS, # data</td>
<td>Or immediate to BUS</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

## Registers

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>INC R</td>
<td>Increment register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INC @R</td>
<td>Increment data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DEC R</td>
<td>Decrement register</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

## Branch

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP addr</td>
<td>Jump unconditional</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JMPP @A</td>
<td>Jump indirect</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ R, addr</td>
<td>Decrement register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC addr</td>
<td>Jump on carry = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JZ addr</td>
<td>Jump on carry = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNZ addr</td>
<td>Jump on A zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JT0 addr</td>
<td>Jump on TO = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT0 addr</td>
<td>Jump on TO = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JT1 addr</td>
<td>Jump on T1 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNT1 addr</td>
<td>Jump on T1 = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF0 addr</td>
<td>Jump on F0 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JF1 addr</td>
<td>Jump on F1 = 1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JTF addr</td>
<td>Jump on timer flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNI addr</td>
<td>Jump on INT = 0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JBb addr</td>
<td>Jump on accumulator bit</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

## Subroutine

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL addr</td>
<td>Jump to subroutine</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td>Return</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RETR</td>
<td>Return and restore</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

## Flags

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>Clear carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F0</td>
<td>Clear flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F0</td>
<td>Complement flag 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR F1</td>
<td>Clear flag 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL F1</td>
<td>Complement flag 1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 2. Instruction Set (Continued)

#### Data Moves

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, R</td>
<td>Move register to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, @R</td>
<td>Move data memory to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, # data</td>
<td>Move immediate to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV R, A</td>
<td>Move A to register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV @R, A</td>
<td>Move A to data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV R, # data</td>
<td>Move immediate to register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV @R, # data</td>
<td>Move immediate to data memory</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV A, PSW</td>
<td>Move PSW to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV PSW, A</td>
<td>Move A to PSW</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, R</td>
<td>Exchange A and register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A, @R</td>
<td>Exchange A and data memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCHD A, @R</td>
<td>Exchange nibble of A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOVX A, @R</td>
<td>Move external data memory to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVX @R, A</td>
<td>Move A to external data memory</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVPA, @A</td>
<td>Move to A from current page</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVPA3 A, @A</td>
<td>Move to A from page 3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Timer/Counter

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, T</td>
<td>Read timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV T, A</td>
<td>Load timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT T</td>
<td>Start timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STRT CNT</td>
<td>Start counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STOP TCNT</td>
<td>Stop timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EN TCNTI</td>
<td>Enable timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS TCNTI</td>
<td>Disable timer/counter</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Control

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN I</td>
<td>Enable external interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIS I</td>
<td>Disable external interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB0</td>
<td>Select register bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL RB1</td>
<td>Select register bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL MB0</td>
<td>Select memory bank 0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SEL MB1</td>
<td>Select memory bank 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*ENT0 CLK</td>
<td>Enable clock output on T0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Bytes</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>No operation</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IDL</td>
<td>Select Idle Operation</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### ABSOLUTE MAXIMUM RATINGS*

- Ambient Temperature Under Bias: 0°C to 70°C
- Storage Temperature: -65°C to +150°C
- Voltage On Any Pin With Respect to Ground: -0.5V to VCC + 1V
- Maximum Voltage On Any Pin With Respect to Ground: 7V
- Power Dissipation: 1.0 Watt

*NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.
D.C. CHARACTERISTICS: \((T_A = 0^\circ C \text{ to } 70^\circ C; V_{CC} = V_{DD} = 5V \pm 20\%; |V_{CC} - V_{DD}| \leq 1.5V; V_{SS} = 0V)\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V_{IL})</td>
<td>Input Low Voltage (All Except X1, RESET)</td>
<td>-0.5 (\text{to} ) 18 (V_{CC})</td>
<td>V</td>
</tr>
<tr>
<td>(V_{IL1})</td>
<td>Input Low Voltage X1, RESET</td>
<td>-5 (\text{to} ) 13 (V_{CC})</td>
<td>V</td>
</tr>
<tr>
<td>(V_{IH})</td>
<td>Input High Voltage (All Except XTAL1, RESET)</td>
<td>0.2 (V_{CC}) + 1.2</td>
<td>V</td>
</tr>
<tr>
<td>(V_{IH1})</td>
<td>Input High Voltage (X1, RESET)</td>
<td>0.7 (V_{CC})</td>
<td>V</td>
</tr>
<tr>
<td>(V_{OL})</td>
<td>Output Low Voltage (BUS)</td>
<td>0.6 (\text{to} ) 1.8 (mA)</td>
<td>(I_{OL} = 2.0\ mA)</td>
</tr>
<tr>
<td>(V_{OL1})</td>
<td>Output Low Voltage (RD, WR, PSEN, ALE)</td>
<td>0.6 (\text{to} ) 1.6 (mA)</td>
<td>(I_{OL} = 1.0\ mA)</td>
</tr>
<tr>
<td>(V_{OL2})</td>
<td>Output Low Voltage (PROG)</td>
<td>0.6 (\text{to} ) 1.6 (mA)</td>
<td>(I_{OL} = 1.0\ mA)</td>
</tr>
<tr>
<td>(V_{OL3})</td>
<td>Output Low Voltage (All Other Outputs)</td>
<td>0.6 (\text{to} ) 1.6 (mA)</td>
<td>(I_{OL} = 1.0\ mA)</td>
</tr>
<tr>
<td>(V_{OH})</td>
<td>Output High Voltage (BUS)</td>
<td>2.4 (\text{to} ) 3.0 (\text{V})</td>
<td>V</td>
</tr>
<tr>
<td>(V_{OH1})</td>
<td>Output High Voltage (RD, WR, PSEN; ALE)</td>
<td>2.4 (\text{to} ) 3.0 (\text{V})</td>
<td>V</td>
</tr>
<tr>
<td>(V_{OH2})</td>
<td>Output High Voltage (All Other Outputs)</td>
<td>2.4 (\text{to} ) 3.0 (\text{V})</td>
<td>V</td>
</tr>
<tr>
<td>(I_{L1})</td>
<td>Input Leakage Current (T1, INT, EA)</td>
<td>5 (\mu A) (\text{to} ) 100 (\mu A)</td>
<td>V</td>
</tr>
<tr>
<td>(I_{L11})</td>
<td>Input Leakage Current (P10–P17, P20–P27, SS)</td>
<td>500 (\mu A) (\text{to} ) 1000 (\mu A)</td>
<td>V</td>
</tr>
<tr>
<td>(I_{L0})</td>
<td>Output Leakage Current (BUS, TO) (High Impedance State)</td>
<td>5 (\mu A) (\text{to} ) 100 (\mu A)</td>
<td>V</td>
</tr>
<tr>
<td>(I_{LR})</td>
<td>Input Leakage Current (RESET)</td>
<td>10 (\mu A) (\text{to} ) 300 (\mu A)</td>
<td>V</td>
</tr>
<tr>
<td>(I_{PD})</td>
<td>Power Down Standby Current</td>
<td>2 (\mu A)</td>
<td>V</td>
</tr>
</tbody>
</table>

\[\begin{array}{|c|c|c|c|}
\hline
\text{ICC Active Current (mA)} & \text{ICC Idle Current (mA)} \\
\hline
\text{V}_{CC} & 4V & 5V & 6V & \text{V}_{CC} & 4V & 5V & 6V \\
\hline
1 MHz & 2.5 & 3.3 & 4.0 & 1 MHz & 1.7 & 2.0 & 2.2 \\
6 MHz & 5 & 6.8 & 8.5 & 6 MHz & 2 & 3 & 4 \\
11 MHz & 9 & 12 & 15 & 11 MHz & 3.5 & 4.8 & 6 \\
\hline
\end{array}\]

**Absolute Maximum Unloaded Current**

**ICC Test Conditions:**

**ICC Active**
All outputs disconnected
T1, INT, SS, T0 connected to HIGH (\(V_{IH}\))
EA, RST connected to LOW (\(V_{IL}\))
XTAL1 External Drive
- Rise Time = 10 ns, Fall Time = 10 ns
XTAL2 External Drive
- Rise Time = 10 ns, Fall Time = 10 ns
XTAL2 No connection
\(V_{IH} = V_{CC} - 0.5V\)
\(V_{IL} = V_{SS} + 0.5V\)

**ICC Idle**
All outputs disconnected
XTAL1 External Drive
- Rise Time = 10 ns, Fall Time = 10 ns
XTAL2 No connection
\(V_{IH} = V_{CC} - 0.5V\)
\(V_{IL} = V_{SS} + 0.5V\)
A.C. CHARACTERISTICS: \((T_A = 0^\circ \text{C} \text{ to } 70^\circ \text{C}; V_{CC} = V_{DD} = 5V \pm 20\%; |V_{CC} - V_{DD}| \leq 1.5V; V_{SS} = 0V)\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>(f(t))</th>
<th>11 MHz</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1)</td>
<td>Clock Period</td>
<td>1/(xtal) freq</td>
<td>90.9</td>
<td>1000</td>
</tr>
<tr>
<td>(t_{LL})</td>
<td>ALE Pulse Width</td>
<td>3.5t-170</td>
<td>150</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{AL})</td>
<td>Addr Setup to ALE</td>
<td>2t-110</td>
<td>70</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{LA})</td>
<td>Addr Hold from ALE</td>
<td>t-40</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{CC1})</td>
<td>Control Pulse Width (RD, WR)</td>
<td>7.5t-200</td>
<td>480</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{CC2})</td>
<td>Control Pulse Width (PSEN)</td>
<td>6t-200</td>
<td>350</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{DW})</td>
<td>Data Setup before WR</td>
<td>6.5t-200</td>
<td>390</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{WR})</td>
<td>Data Hold after WR</td>
<td>t-50</td>
<td>40</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{DR})</td>
<td>Data Hold (RD, PSEN)</td>
<td>1.5t-30</td>
<td>0</td>
<td>110</td>
</tr>
<tr>
<td>(t_{RD1})</td>
<td>RD to Data in</td>
<td>6t-170</td>
<td>350</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{RD2})</td>
<td>PSEN to Data in</td>
<td>4.5t-170</td>
<td>190</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{AW})</td>
<td>Addr Setup to WR</td>
<td>5t-150</td>
<td>300</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{AD1})</td>
<td>Addr Setup to Data (RD)</td>
<td>10.5t-220</td>
<td>730</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{AD2})</td>
<td>Addr Setup to Data (PSEN)</td>
<td>7.5t-220</td>
<td>460</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{AFC1})</td>
<td>Addr Float to RD, WR</td>
<td>2t-40</td>
<td>140</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{AFC2})</td>
<td>Addr Float to PSEN</td>
<td>.5t-40</td>
<td>10</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{LAFC1})</td>
<td>ALE to Control (RD, WR)</td>
<td>3t-75</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{LAFC2})</td>
<td>ALE to Control (PSEN)</td>
<td>1.5t-75</td>
<td>60</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{CA1})</td>
<td>Control to ALE (RD, WR, PROG)</td>
<td>t-65</td>
<td>25</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{CA2})</td>
<td>Control to ALE (PSEN)</td>
<td>4t-70</td>
<td>290</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{CP})</td>
<td>Port Control Setup to PROG</td>
<td>1.5t-80</td>
<td>50</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{PC})</td>
<td>Port Control Hold to PROG</td>
<td>4t-260</td>
<td>100</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{PR})</td>
<td>PROG to P2 Input Valid</td>
<td>8.5t-120</td>
<td>650</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{PF})</td>
<td>Input Data Hold from PROG</td>
<td>1.5t</td>
<td>140</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{DP})</td>
<td>Output Data Setup</td>
<td>6t-290</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{PD})</td>
<td>Output Data Hold</td>
<td>1.5t-90</td>
<td>40</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{PP})</td>
<td>PROG Pulse Width</td>
<td>10.5t-250</td>
<td>700</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{LP})</td>
<td>Port 2 I/O Setup to ALE</td>
<td>4t-200</td>
<td>160</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{LP})</td>
<td>Port 2 I/O Hold to ALE</td>
<td>1.5t-120</td>
<td>15</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{PV})</td>
<td>Port Output from ALE</td>
<td>4.5t+100</td>
<td>510</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{OPRR})</td>
<td>T0 Rep Rate</td>
<td>3t</td>
<td>270</td>
<td>ns</td>
</tr>
<tr>
<td>(t_{CY})</td>
<td>Cycle Time</td>
<td>15t</td>
<td>1.36</td>
<td>15.0</td>
</tr>
</tbody>
</table>

Notes:
1. Control Outputs CL = 80pF
   BUS Outputs CL = 150pF
2. BUS High Impedance
   Load 20pF
3. \(f(t)\) assumes 50\% duty cycle on X1, X2. Max.
   clock period is for a 1 MHz crystal input.
WAVEFORMS

Instruction Fetch From Program Memory

Read From External Data Memory

Write To External Data Memory

Input And Output For A.C. Tests

PORT 1/PORT 2 TIMING

A.C. testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

1ST CYCLE

2ND CYCLE
SUGGESTED ROM VERIFICATION ALGORITHM FOR CHMOS DEVICES ONLY

INITIAL ROM DUMP CYCLE

- ALE
- +12V
- EA

SUBSEQUENT ROM DUMP CYCLES

- ADDRESS (INPUT)
- ROM DATA (OUTPUT)
- ADDRESS (INPUT)
- ROM DATA (OUTPUT)

P20–P23

<table>
<thead>
<tr>
<th>C48</th>
<th>C49</th>
<th>C50</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A10</td>
<td>0</td>
<td>ADDR</td>
<td>ADDR</td>
</tr>
<tr>
<td>A11</td>
<td>0</td>
<td>0</td>
<td>ADDR</td>
</tr>
</tbody>
</table>

VCC = VDD = +5V

VSS = 0V
CHAPTER 16
THE RUPI™-44 FAMILY:
MICROCONTROLLER WITH ON-CHIP
COMMUNICATION CONTROLLER

16.0 INTRODUCTION
The RUPI-44 family is designed for applications requiring local intelligence at remote nodes, and communication capability among these distributed nodes. The RUPI-44 integrates onto a single chip Intel's highest performance microcontroller, the 8051-core, with an intelligent and high performance Serial communication controller, called the Serial Interface Unit, or SIU. See Figure 16-1. This dual controller architecture allows complex control and high speed data communication functions to be realized cost effectively.

The RUPI-44 family consists of three pin compatible parts:
- 8344—8051 Microcontroller with SIU
- 8044—An 8344 with 4K bytes of on-chip ROM program memory.
- 8744—An 8344 with 4K bytes of on-chip EPROM program memory.

16.1 ARCHITECTURE OVERVIEW
The 8044's dual controller architecture enables the RUPI to perform complex control tasks and high speed communication in a distributed network environment.

The 8044 microcontroller is the 8051-core, and maintains complete software compatibility with it. The microcontroller contains a powerful CPU with on-chip peripherals, making it capable of serving sophisticated real-time control applications such as instrumentation, industrial control, and intelligent computer peripherals. The microcontroller features on-chip peripherals such as two 16-bit timer/counters and 5 source interrupt capability with programmable priority levels. The microcontroller's high performance CPU executes most instructions in 1 microsecond, and can perform an $8 \times 8$ multiply in 4 microseconds. The CPU features a Boolean processor that can perform operations on 256 directly addressable bits. 192 bytes of on-chip data RAM can be extended to 64K bytes externally. 4K bytes of on-chip program ROM can be extended to 64K bytes externally. The CPU and SIU run concurrently. See Figure 16-2.

The SIU is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4Mbps, externally clocked, and 375K bps self clocked (i.e., the data clock is recovered by an on-chip digital phase locked loop). SIU hardware supports the HDLC/SDLC protocol: zero bit insertion/deletion, address recognition, cyclic redundancy check, and frame number sequence check are automatically performed.

The SIU's Auto mode greatly reduces communication software overhead. The AUTO mode supports the SDLC Normal Response Mode, by performing secondary station responses in hardware without any CPU involvement. The Auto mode's interrupt control and frame sequence numbering capability eliminates software overhead normally required in conventional systems. By using the Auto mode, the CPU is free to concentrate on real time control of the application.

![Figure 16-1. RUPI™-44 Dual Controller Architecture](image-url)
Figure 16-2: Simplified 8044 Block Diagram
16.2 THE HDLC/SDLC PROTOCOLS

16.2.1 HDLC/SDLC Advantages over Async

The High Level Data Link Control, HDLC, is a standard communication link control established by the International Standards Organization (ISO). SDLC is a subset of HDLC.

HDLC and SDLC are both well recognized standard serial protocols. The Synchronous Data Link Control, SDLC, is an IBM standard communication protocol. IBM originally developed SDLC to provide efficient, reliable and simple communication between terminals and computers.

The major advantages of SDLC/HDLC over Asynchronous communications protocol (Async):

- **SIMPLE:** Data Transparency
- **EFFICIENT:** Well Defined Message-Level Operation
- **RELIABLE:** Frame Check Sequence and Frame Numbering

The SDLC reduces system complexity. HDLC/SDLC are “data transparent” protocols. Data transparency means that an arbitrary data stream can be sent without concern that some of data could be mistaken for a protocol controller. Data transparency relieves the communication controller having to detect special characters.

SDLC/HDLC provides more data throughput than Async. SDLC/HDLC runs at Message-level Operation which transmits multiple bytes within the frame. Whereas Async is based on character-level operation. Async transmits or receives a character at a time. Since Async requires start and stop bits in every transmission, there is a considerable waste of overhead compared to SDLC/HDLC.

![Figure 16-3. RUPI™-44 Supported Network Configurations](image-url)
Due to SDLC/HDLC’s well delineated field (see Figure 16-4) the CPU does not have to interpret character by character to determine control field and information field. In the case of Async, CPU must look at each character to interpret what it means. The practical advantage of such feature is straightforward use of DMA for information transfer.

In addition, SDLC/HDLC further improves data throughput using implied Acknowledgement of transferred information. A station using SDLC/HDLC may acknowledge previously received information while transmitting different information in the same frame. In addition, up to 7 messages may be outstanding before an acknowledgement is required.

The HDLC/SDLC protocol can be used to realize reliable data links. Reliable data transmission is ensured at the bit level by sending a frame check sequence, cyclic redundancy checking, within the frame. Reliable frame transmission is ensured by sending a frame number identification with each frame. This means that a receiver can sequentially count received frames and at any time infer what the number of the next frame to be received should be. More important, it provides a means for the receiver to identify to the sender some particular frame that it wishes to have resent because of errors.

16.2.2 HDLC/SDLC Networks

In both the HDLC and SDLC line protocols a (Master) primary station controls the overall network (data link) and issues commands to the secondary (Slave) stations. The latter complies with instructions and responds by sending appropriate responses. Whenever a transmitting station must end transmission prematurely, it sends an abort character. Upon detecting an abort character, a receiving station ignores the transmission block called a frame.

RUPI-44 supported HDLC/SDLC network configurations are point-to-point (half duplex), and loop. In the loop configuration the stations themselves act as repeaters, so that long links can be easily realized, see Figure 16-3.

16.2.3 Frames

An HDLC/SDLC frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags—opening and closing flags. An address field is 8 bits wide in SDLC, extendable to 2 or more bytes in HDLC. The control field is also 8 bits wide, extendable to two bytes in HDLC. The SDLC data field or information field may be any number of bytes. The HDLC data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags. See Figure 16-4.

In HDLC and SDLC are three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Non-sequenced Frame is used for initialization and control of the secondary stations.

For a more detailed discussion of higher level protocol functions interested readers may refer to the references listed in Section 16.2.6.

16.2.4 Zero Bit Insertion

In data communications, it is desirable to transmit data which can be of arbitrary content. Arbitrary data transmission requires that the data field cannot contain characters which are defined to assist the transmission protocol (like opening flag in HDLC/SDLC communications). This property is referred to as “data transparency”. In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion (ZBI).

The flag has a unique binary bit pattern: 01111110 (7E HEX). To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique

<table>
<thead>
<tr>
<th>OPENING FLAG</th>
<th>ADDRESS FIELD</th>
<th>CONTROL FIELD</th>
<th>INFORMATION FIELD</th>
<th>FRAME CHECK SEQUENCE (FCS)</th>
<th>CLOSING FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111110</td>
<td>8 BITS</td>
<td>8 BITS</td>
<td>VARIABLE LENGTH (ONLY IN I FRAMES)</td>
<td>16 BITS</td>
<td>01111110</td>
</tr>
</tbody>
</table>

Figure 16-4. Frame Format
called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8044 performs zero bit insertion and deletion automatically.

16.2.5 Non-return to Zero Inverted (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC/SDLC protocol. It allows HDLC/SDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop (DPLL) techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while an 0 causes a change of state. NRZI coding ensures that an active data line will have a transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for the 8044's on-chip DPLL to recover a receive clock (from received data) synchronized to the received data and at the same time ensure data transparency.

16.2.6 References

1. IBM Synchronous Data Link Control General Information GA27-3093-2 File No. GENL-09.
3. IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA27-3098-0.

16.3 RUPI™-44 DESIGN SUPPORT

16.3.1 Design Tool Support

A critical design consideration is time to market. Intel provides a sophisticated set of design tools to speed hardware and software development time of 8044 based products. These include ICE-44, ASM-51, PL/M-51, and EMV-44.

Figure 16-5. RUPI™-44 Development Support Configuration Intellec® System, ICE™-44 Buffer Box, and ICE-44 Module Plugged into a User Prototype Board.

A primary tool is the 8044 In Circuit Emulator, called ICE-44. See Figure 16-5. In conjunction with Intel's Intellec® Microprocessor Development System, the ICE-44 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-44 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed.

The ICE-44 emulator assists four stages of development:

1) Software Debugging

It can be operated without being connected to the user's system before any of the user's hardware is available. In this stage ICE-44 debugging capabilities can be used in conjunction with the Intellec text edi-
tor and 8044 macroassembler to facilitate program development.

2) Hardware Development

The ICE-44 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware, including the time-critical SDLC serial port, parallel port, and timer interfaces.

3) System Integration

Integration of software and hardware can begin when any functional element of the user system hardware is connected to the 8044 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is system tested in real-time operation as it becomes available.

4) System Test

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-44 module is then used for real-time emulation of the 8044 to debug the system as a completed unit.

The final product verification test may be performed using the 8744 EPROM version of the 8044 microcomputer. Thus, the ICE-44 module provides the user with the ability to debug a prototype or production system at any stage in its development.

A conversion kit, ICE-44 CON, is available to upgrade an ICE-51 module to ICE-44.

Intel's ASM-51 Assembler supports the 8044 special function registers and assembly program development. PL/M-51 provides designers with a high level language for the 8044. Programming in PL/M can greatly reduce development time, and ensure quick time to market.

These tools have recently been expanded with the addition of the EMV-44CON. This conversion kit allows you to convert an EMV-51 into an EMV-44 emulation vehicle. The resultant low cost emulator is design for use with an iPDS Personal Development System, which also supports the ASM-51 assembler and PL/M-51. See Figure 16-6.

![Figure 16-6. RUPI-44 iPDS Personal Development System, EMV-44 Buffer Box, and EMV-44 Module Plugged into a User Prototype Board.](image)

Emulation support is similar to the ICE-44 with support for Software and Hardware Development, System Integration, and System Test. The iPDS's rugged portability and ease of use also make it an ideal system for production tests and field service of your finished design. In addition, the iPDS offers EPROM programming module for the 8744, and direct communications with the 8044-based BITBUS via an optional iSBX344 distributed control module.

16.3.2 8051 Workshop

Intel provides 8051 training to its customers through the 5-day 8051 workshop. Familiarity with the 8051 and 8044 is achieved through a combination of lecture and laboratory exercises.

For designers not familiar with the 8051, the workshop is an effective way to become proficient with the 8051 architecture and capabilities.
CHAPTER 17
8044 ARCHITECTURE

17.0 GENERAL

The 8044 is based on the 8051 core. The 8044 replaces the 8051's serial port with an intelligent HDLC/SDLC controller called the Serial Interface or SIU. Thus the differences between the two result from the 8044's increased on-chip RAM (192 bytes) and additional special function registers necessary to control the SIU. Aside from the increased memory, the SIU itself, and differences in 5 pins (for the serial port), the 8044 and 8051 are compatible.

This chapter describes the differences between the 8044 and 8051. Information pertaining to the 8051 core, eg. instruction set, port operation, EPROM programming, etc. is located in the 8051 sections of this manual.

A block diagram of the 8044 is shown in Figure 17-1.

The pinpoint is shown on the inside front cover.

17.1 MEMORY ORGANIZATION OVERVIEW

The 8044 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM.

If the EA pin is held high, the 8044 executes out of internal ROM unless the Program Counter exceeds 0FFH. Fetches from locations 1000H through FFFFH are directed to external Program Memory.

If the EA pin is held low, the 8044 fetches all instructions from external Program Memory.

The Data Memory consists of 192 bytes of on-chip RAM, plus 35 Special Function Registers, in addition to which the device is capable of accessing up to 64K bytes of external data memory.

The Program Memory uses 16-bit addresses. The external Data Memory can use either 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses, which provide a 256-location address space. The lower 192 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space.

The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the “working registers” of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus, a large number of instructions are one-byte instructions.

The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one-bit (Boolean) processing. This bit-addressing capability is an important feature of the 8044. In addition to the 128 individually addressable bits in RAM, twelve of the Special Function Registers also have individually addressable bits.

A memory map is shown in Figure 17-2.

17.1.1 Special Function Registers

The Special Function Registers are as follows:

- ACC Accumulator (A Register)
- B B Register
- PSW Program Status Word
- SP Stack Pointer
- DPTR Data Pointer (consisting of DPH
  AND DPL)
- P0 Port 0
- P1 Port 1
- P2 Port 2
- P3 Port 3
- IP Interrupt Priority
- IE Interrupt Enable
- TMOD Timer/Counter Mode
- TCON Timer/Counter Control
- TH0 Timer/Counter 0 (high byte)
- TL0 Timer/Counter 0 (low byte)
- TH1 Timer/Counter 1 (high byte)
- TL1 Timer/Counter 1 (low byte)
- SMD Serial Mode
- STS Status/Command
- NSNR Send/Receive Count
- STAD Station Address
- TBS Transmit Buffer Start Address
- TBL Transmit Buffer Length
- TCB Transmit Control Byte
- RBS Receive Buffer Start Address
- RBL Receive Buffer Length
- RFL Received Field Length
- RCB Received Control Byte
- DMA CNT DMA Count
- FIFO FIFO (three bytes)
- SIUST SIU State Counter
- PCON Power Control

The registers marked with * are both byte- and bit-addressable.
Stack Pointer

The Stack Pointer is 8 bits wide. The stack can reside anywhere in the 192 bytes of on-chip RAM. When the 8044 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL, the stack pointer is incremented before data is stored, so the stack would begin at location 08H.

17.1.2 Interrupt Control Registers

The Interrupt Request Flags are as listed below:

<table>
<thead>
<tr>
<th>Source</th>
<th>Request Flag</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Interrupt</td>
<td>INTO, if IT0 = 0</td>
<td>P3.2</td>
</tr>
<tr>
<td></td>
<td>IT0, if IT0 = 1</td>
<td>TCON.1</td>
</tr>
<tr>
<td>Timer 0 Overflow</td>
<td>TF0</td>
<td>TCON.5</td>
</tr>
<tr>
<td>External Interrupt</td>
<td>INT1, if IT1 = 0</td>
<td>P3.3</td>
</tr>
<tr>
<td></td>
<td>IE1, if IT1 = 1</td>
<td>TCON.3</td>
</tr>
<tr>
<td>Timer 1 Overflow</td>
<td>TF1</td>
<td>TCON.7</td>
</tr>
<tr>
<td>Serial Interface Unit SI</td>
<td>STS.4</td>
<td></td>
</tr>
</tbody>
</table>

External Interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All the interrupt flags can be set or cleared by software, with the same effect as by hardware.

The Enable and Priority Control Registers are shown below. All of these control bits are set or cleared by software. All are cleared by reset.

IE: Interrupt Enable Register (bit-addressable)

Bit:  7  6  5  4  3  2  1  0

where:

- EA disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
enables or disables the Serial Interface Unit interrupt. If ES = 0, the Serial Interface Unit interrupt is disabled.

- ET1 enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.

- EX1 enables or disables the External Interrupt 1. If EX1 = 0, the External Interrupt 1 is disabled.

- ET0 enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.

### 17.2 Memory Organization Details

In the 8044 family, the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 18-2 are the:

- 64K-byte Program Memory address space
- 64K-byte External Data Memory address space
- 320-byte Internal Data Memory address space

The 16-bit Program Counter register provides the 8044 with its 64K addressing capabilities. The Program Counter allows the user to execute calls and branches to any location within the Program Memory space. There are no instructions that permit program execution to move from the Program Memory space to any of the data memory spaces.

In the 8044 and 8744, the lower 4K of the 64K Program Memory address space is filled by internal ROM and EPROM, respectively. By tying the EA pin high, the processor can be forced to fetch from the internal ROM/EPROM for Program Memory addresses 0 through 4K. Bus expansion for accessing Program Memory beyond 4K is automatic since external instruction fetches occur automatically when the Program Counter increases above 4095. If the EA pin is tied low, all Program Memory fetches are from external memory. The execution speed of the 8044 is the same regardless of whether fetches are from internal or external Program Memory. If all program storage is on-chip, byte location 4095 should be left vacant to prevent an undesired prefetch from external Program Memory address 4096.

Certain locations in Program Memory are reserved for specific programs. Locations 0000 through 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 through 0042 are reserved for the five interrupt-request service programs. Each resource that can request an interrupt requires that its service program be stored at its reserved location.

The 64K-byte External Data Memory address space is automatically accessed when the MOVX instruction is executed.

Functionally, the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 17-3.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently, locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

The stack depth is limited only by the available Internal Data RAM, thanks to an 8-bit reloadable Stack Pointer. The stack is used for storing the Program Counter during subroutine calls and may be used for passing parameters. Any byte of Internal Data RAM or Special Function Register accessible through Direct Addressing can be pushed/popped.
The Special Function Register address space is 128 to 255. All registers except the Program Counter and the four 8-Register Banks reside here. Memory mapping the Special Function Registers allows them to be accessed as easily as internal RAM. As such, they can be operated on by most instructions. In the overlapping memory space (address 128-191), indirect addressing is used to access RAM, and direct addressing is used to access the SFR's. The SFR's at addresses 192-255 are also accessed using direct addressing. The Special Function Registers are listed in Figure 17-4. Their mapping in the Special Function Register address space is shown in Figures 17-5 and 17-6.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 192-255) nor a Special Function Register exists will access data of indeterminable value.

Architecturally, each memory space is a linear sequence of 8-bit wide bytes. By Intel convention the storage of multi-byte address and data operands in program and data memories is the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte X, the most significant bit is represented by X.7 while the least significant bit is X.0.

Any deviation from these conventions will be explicitly stated in the text.

### 17.2.1 Operand Addressing

There are five methods of addressing source operands. They are Register Addressing, Direct Addressing, Register-Indirect Addressing, Immediate Addressing and Base-Register-plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8044 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8044 with its 21 addressing modes.

Most instructions have a “destination, source” field that specifies the data type, addressing methods and operands involved. For operations other than moves, the destination operand is also a source operand. For example, in “subtract-with-borrow A, #5” the A register receives the result of the value in register A minus 5, minus C.

Most operations involve operands that are located in Internal Data Memory. The selection of the Program Memory space or External Data Memory space for a

---

**Figure 17-3. Internal Data Memory Address Space**

**Figure 17-4. Special Function Registers**
second operand is determined by the operation mnemonic unless it is an immediate operand. The subset of the Internal Data Memory being addressed is determined by the addressing method and address value. For example, the Special Function Registers can be accessed only through Direct Addressing with an address of 128–255. A summary of the operand addressing methods is shown in Figure 17-6. The following paragraphs describe the five addressing methods.

### 17.2.2 Register Addressing

Register Addressing permits access to the eight registers (R7–R0) of the selected Register Bank (RB). One of the four 8-Register Banks is selected by a two-bit field in the PSW. The registers may also be accessed through Direct Addressing and Register-Indirect Addressing, since the four Register Banks are mapped into the lowest 32 bytes of Internal Data RAM as shown in Figures 17-9 and 17-10. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

---

**Figure 17-5. Mapping of Special Function Registers**

<table>
<thead>
<tr>
<th>REGISTER NAMES</th>
<th>SYMBOLIC ADDRESS</th>
<th>BIT ADDRESS</th>
<th>BYTE ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B REGISTER</td>
<td>B</td>
<td>247 through 240</td>
<td>240 (FOH)</td>
</tr>
<tr>
<td>ACCUMULATOR</td>
<td>ACC</td>
<td>231 through 224</td>
<td>224 (EDH)</td>
</tr>
<tr>
<td>*THREE BYTE FIFO</td>
<td>FIFO</td>
<td>215 through 216</td>
<td>219 (EBH)</td>
</tr>
<tr>
<td>TRANSMIT BUFFER START</td>
<td>TBS</td>
<td>205 through 208</td>
<td>208 (DCH)</td>
</tr>
<tr>
<td>TRANSMIT BUFFER LENGTH</td>
<td>TBL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSMIT CONTROL BYTE</td>
<td>TCB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*SIU STATE COUNTER</td>
<td>SIUST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEND COUNT RECEIVE COUNT</td>
<td>NSNR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAM STATUS WORD</td>
<td>PSW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*DMA COUNT</td>
<td>DMA CNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATION ADDRESS</td>
<td>STAD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEIVE FIELD LENGTH</td>
<td>RFL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEIVE BUFFER START</td>
<td>RBS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEIVE BUFFER LENGTH</td>
<td>RBL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECEIVE CONTROL BYTE</td>
<td>RCb</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERIAL MODE</td>
<td>SMD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STATUS REGISTER</td>
<td>STS</td>
<td>207 through 200</td>
<td>200 (CFH)</td>
</tr>
<tr>
<td>INTERRUPT PRIORITY CONTROL</td>
<td>IP</td>
<td>191 through 184</td>
<td>184 (BFH)</td>
</tr>
<tr>
<td>PORT 3</td>
<td>P3</td>
<td>183 through 176</td>
<td>176 (B0H)</td>
</tr>
<tr>
<td>INTERRUPT ENABLE CONTROL</td>
<td>IE</td>
<td>175 through 168</td>
<td>168 (A0H)</td>
</tr>
<tr>
<td>PORT 2</td>
<td>P2</td>
<td>167 through 160</td>
<td>160 (A0H)</td>
</tr>
<tr>
<td>PORT 1</td>
<td>P1</td>
<td>151 through 144</td>
<td>144 (90H)</td>
</tr>
<tr>
<td>TIMER HIGH 1</td>
<td>TH1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER HIGH 0</td>
<td>TH0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER LOW 1</td>
<td>TL1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER LOW 0</td>
<td>TL0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER MODE</td>
<td>TMOD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMER CONTROL</td>
<td>TCON</td>
<td>143 through 136</td>
<td>136 (88H)</td>
</tr>
<tr>
<td>DATA POINTER HIGH</td>
<td>DPH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA POINTER LOW</td>
<td>DPL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STACK POINTER</td>
<td>SP</td>
<td>135 through 128</td>
<td>128 (B0H)</td>
</tr>
</tbody>
</table>

*SFR's CONTAINING DIRECT ADDRESSABLE BITS

- SFR's containing direct addressable bits
- *ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

---

**Figure 17-6. Mapping of Special Function Registers**
### Figure 17-7. Special Function Register Bit Address

<table>
<thead>
<tr>
<th>Direct Byte Address (MSB)</th>
<th>Bit Address (LSB)</th>
<th>Hardware Register Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>F7 F6 F5 F4 F3 F2 F1 F0</td>
<td>8</td>
</tr>
<tr>
<td>224</td>
<td>E7 E6 E5 E4 E3 E2 E1 E0</td>
<td>ACC</td>
</tr>
<tr>
<td>NS2 NS1 NS0 SE5 SE4 SE3 SE2 SE1 SE0</td>
<td>NSNR</td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>DF DE DC DB DA D9 D8</td>
<td>CY AC RS1 RSO OV P</td>
</tr>
<tr>
<td>204</td>
<td>D7 D6 D5 D4 D3 D2 D1 D0</td>
<td>TBF RE RTS SI BV CPB AM RBP</td>
</tr>
<tr>
<td>200</td>
<td>CF CE CD CC CB CA CB</td>
<td>STS</td>
</tr>
<tr>
<td>184</td>
<td>- - - - BC BB BA BA B8</td>
<td>PS PT1 PX1 PT0 PX0</td>
</tr>
<tr>
<td>176</td>
<td>B7 B6 B5 B4 B3 B2 B1 B0</td>
<td>P3</td>
</tr>
<tr>
<td>168</td>
<td>EA E5 ET1 EX1 EX0</td>
<td>P2</td>
</tr>
<tr>
<td>160</td>
<td>A7 A6 A5 A4 A3 A2 A1 A0</td>
<td>P1</td>
</tr>
<tr>
<td>144</td>
<td>97 96 95 94 93 92 91 90</td>
<td>TF1 TR1 TFO TRO IE1 IT1 IEO ITO</td>
</tr>
<tr>
<td>136</td>
<td>BF BE BD BC BB BA BA B8</td>
<td>TCON</td>
</tr>
<tr>
<td>128</td>
<td>87 86 85 84 83 82 81 80</td>
<td>P0</td>
</tr>
</tbody>
</table>

### Figure 17-8. Operand Addressing Methods

- **Register Addressing**
  - R7-R0
  - A, B, C (bit), AB (two bytes), DPTR (double byte)
- **Direct Addressing**
  - Lower 128 bytes of Internal Data RAM
  - Special Function Registers
- **Register-Indirect Addressing**
  - Internal Data RAM (@R1, @R0, @SP (PUSH and POP only))
  - Least Significant Nibbles in Internal Data RAM (@R1, @R0)
  - External Data Memory (@R1, @R0, @DPTR)
- **Immediate Addressing**
  - Program Memory (in-code constant)
- **Base-Register-plus Index-Register-Indirect Addressing**
  - Program Memory (@DPTR + A, @PC + A)

### Figure 17-9. RAM Bit Addresses

### Figure 17-10. Addressing Operands in Internal Data Memory
17.2.3 Direct Addressing

Direct Addressing provides the only means of accessing the memory-mapped byte-wide Special Function Registers and memory mapped bits within the Special Function Registers and Internal Data RAM. Direct Addressing of bytes may also be used to access the lower 128 bytes of Internal Data RAM. Direct Addressing of bits gains access to a 128 bit subset of the Internal Data RAM and 128 bit subset of the Special Function Registers as shown in Figures 17-5, 17-6, 17-9, and 17-10.

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or using the content of the Stack Pointer (PUSH and POP only), addresses the Internal Data RAM. Register-Indirect Addressing is also used for accessing the External Data Memory. In this case, either R1 or R0 in the selected Register Bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit Data Pointer may be used for accessing any location within the full 64K external address space.

17.3 RESET

Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) while the oscillator is running. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

<table>
<thead>
<tr>
<th>Register</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>0000H</td>
</tr>
<tr>
<td>A</td>
<td>00H</td>
</tr>
<tr>
<td>B</td>
<td>00H</td>
</tr>
<tr>
<td>PSW</td>
<td>00H</td>
</tr>
<tr>
<td>SP</td>
<td>07H</td>
</tr>
<tr>
<td>DPTR</td>
<td>0000H</td>
</tr>
<tr>
<td>P0 – P3</td>
<td>OFFH</td>
</tr>
<tr>
<td>IP</td>
<td>(XXX00000)</td>
</tr>
<tr>
<td>IE</td>
<td>(0XXX0000)</td>
</tr>
<tr>
<td>TMOD</td>
<td>00H</td>
</tr>
<tr>
<td>TCON</td>
<td>00H</td>
</tr>
<tr>
<td>TH0</td>
<td>00H</td>
</tr>
<tr>
<td>TL0</td>
<td>00H</td>
</tr>
<tr>
<td>TH1</td>
<td>00H</td>
</tr>
<tr>
<td>TL1</td>
<td>00H</td>
</tr>
<tr>
<td>SMD</td>
<td>00H</td>
</tr>
<tr>
<td>STS</td>
<td>00H</td>
</tr>
<tr>
<td>NSNR</td>
<td>00H</td>
</tr>
<tr>
<td>STAD</td>
<td>00H</td>
</tr>
<tr>
<td>TBS</td>
<td>00H</td>
</tr>
<tr>
<td>TBL</td>
<td>00H</td>
</tr>
<tr>
<td>TCB</td>
<td>00H</td>
</tr>
<tr>
<td>RBS</td>
<td>00H</td>
</tr>
<tr>
<td>RBL</td>
<td>00H</td>
</tr>
<tr>
<td>RFL</td>
<td>00H</td>
</tr>
<tr>
<td>RCB</td>
<td>00H</td>
</tr>
<tr>
<td>DMA CNT</td>
<td>00H</td>
</tr>
<tr>
<td>FIFO1</td>
<td>00H</td>
</tr>
<tr>
<td>FIFO2</td>
<td>00H</td>
</tr>
<tr>
<td>FIFO3</td>
<td>00H</td>
</tr>
<tr>
<td>SIUST</td>
<td>01H</td>
</tr>
<tr>
<td>PCON</td>
<td>(0XXXXXXX)</td>
</tr>
</tbody>
</table>

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off (see Power Down Operation.)

17.4 RUPI™-44 FAMILY PIN DESCRIPTION

VSS: Circuit ground potential.

VCC: Supply voltage during programming (of the 8744), verification (of the 8044 or 8744), and normal operation.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during which accesses it activates internal pullups). It also outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification in the 8044 or 8744. Port 1 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Two of the Port 1 pins serve alternate functions, as listed below:

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.6 RTS (Request to Send). In a non-loop configur</td>
<td></td>
</tr>
</tbody>
</table>
P1.7 CTS (Clear to Send). In a non-loop configuration, CTS signals to the 8044 that the receiving station is ready to accept data.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification in the 8044 or 8744. Port 2 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. Port 3 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups. Port 3 pins also serve alternate functions, as listed below:

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RXD (serial input port in loop configuration). I/O (data direction control in non-loop configuration).</td>
</tr>
<tr>
<td>P3.1</td>
<td>TXD (serial output port in loop configuration). DATA input/output pin in non-loop configuration.</td>
</tr>
<tr>
<td>P3.2</td>
<td>INTO (external interrupt)</td>
</tr>
<tr>
<td>P3.3</td>
<td>INTI (external interrupt)</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 (Timer 0 external input)</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 (Timer 1 external input) SCLK (Serial Data Clock Input)</td>
</tr>
<tr>
<td>P3.6</td>
<td>WR (external Data Memory write strobe)</td>
</tr>
<tr>
<td>P3.7</td>
<td>RD (external Data Memory read strobe)</td>
</tr>
</tbody>
</table>

PSEN: Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory two activations of PSEN are skipped during each access to external Data Memory.) PSEN is not activated during fetches from internal Program Memory.

EA/VPP: When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds (OFFH). When EA is held low the CPU executes only out of external Program Memory. In the 8344, EA must be externally wired low. In the 8744, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting amplifier that forms the oscillator. Should be grounded when an external oscillator is used.

XTAL2: Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.
CHAPTER 18
THE 8044 SERIAL INTERFACE UNIT

18.0 SERIAL INTERFACE

The serial interface provides a high-performance communication link. The protocol used for this communication is based on the IBM Synchronous Data Link Control (SDLC). The serial interface also supports a subset of the ISO HDLC (International Standards Organization High-Level Data Link Control) protocol.

The SDLC/HDLC protocols have been accepted as standard protocols for many high-level teleprocessing systems. The serial interface performs many of the functions required to service the data link without intervention from the 8044's own CPU. The programmer is free to concentrate on the 8044's function as a peripheral controller, rather than having to deal with the details of the communication process.

Five pins on the 8044 are involved with the serial interface (refer to Section 12.4, Family Pin Description, for details):

- Pin 7: RTS/P16
- Pin 8: CTS/P17
- Pin 10: I/O/RXD/P30
- Pin 11: DATA/TXD/P31
- Pin 15: SCLK/T1/P35

Figure 18-1 is a functional block diagram of the serial interface unit (SIU). More details on the SIU hardware are given in Section 18.9.

18.1 DATA LINK CONFIGURATIONS

The serial interface is capable of operating in three serial data link configurations:

1) Half-Duplex, point-to-point
2) Half-Duplex, multipoint (with a half-duplex or full-duplex primary)
3) Loop

Figure 18-2 shows these three configurations. The RTS (Request to Send) and CTS (Clear to Send) handshaking signals are available in the point-to-point and multipoint configurations.

18.2 DATA CLOCKING OPTIONS

The serial interface can operate in an externally clocked mode or in a self clocked mode.

Externally Clocked Mode

In the externally clocked mode, a common Serial Data Clock (SCLK on pin 15) synchronizes the serial bit stream. This clock signal may come from the master CPU or primary station, or from an external phase-locked loop local to the 8044. Figure 18-3 illustrates the timing relationships for the serial interface signals when the externally clocked mode is used in point-to-point and multipoint data link configurations.

Incoming data is sampled at the rising edge of SCLK, and outgoing data is shifted out at the falling edge of SCLK. More detailed timing information is given in the 8044 data sheet.

Self Clocked (Asynchronous) Mode

The self clocked mode allows data transfer without a common system data clock. Using an on-chip DPLL (digital phase locked loop) the serial interface recovers the data clock from the data stream itself. The DPLL requires a reference clock equal to either 16 times or 32 times the data rate. This reference clock may be externally supplied or internally generated. When the serial interface generates this clock internally, it uses either the 8044's internal logic clock (half the crystal frequency's PH2) or the "timer 1" overflow. Figure 18-4 shows the serial interface signal timing relationships for the loop configuration, when the unclocked mode is used.

The DPLL monitors the received data in order to derive a data clock that is centered on the received bits. Centering is achieved by detecting all transitions of the received data, and then adjusting the clock transition (in increments of 1/16 bit period) toward the center of the received bit. The DPLL converges to the nominal bit center within eight bit transitions, worst case.

To aid in the phase locked loop capture process, the 8044 has a NRZI (non-return-to-zero inverted) data encoding and decoding option. NRZI coding specifies that a signal does not change state for a transmitted binary 1, but does change state for a binary 0. Using the NRZI coding with zero-bit insertion, it can be guaranteed that an active signal line undergoes a transition at least every six bit times.

18.3 DATA RATES

The maximum data rate in the externally clocked mode is 2.4M bits per second (bps) a half-duplex configuration, and 1.0M in a loop configuration.
Figure 18-2. RUPI-44 Data Link Configurations
In the self clocked mode with an external reference clock, the maximum data rate is 375 Kbps.

In the self clocked mode with an internally generated reference clock, and the 8044 operating with a 12 MHz crystal, the available data rates are 244 bps to 62.5 kbps, 187.5 Kbps and 375 Kbps. For more details see the table in the SMD register description, below.

18.4 OPERATIONAL MODES

The Serial Interface Unit (SIU) can operate in either of two response modes:

1) AUTO mode
2) FLEXIBLE (NON-AUTO) mode

In the AUTO mode, the SIU performs in hardware a subset of the SDLC protocol called the normal response mode. The AUTO mode enables the SIU to recognize and respond to certain kinds of SDLC frames without intervention from the 8044's CPU. AUTO mode provides a faster turnaround time and a simplified software interface, whereas NON-AUTO mode provides a greater flexibility with regard to the kinds of operation permitted.

In AUTO mode, the 8044 can act only as a normal response mode secondary station—that is, it can transmit only when instructed to do so by the primary station. All such AUTO mode responses adhere strictly to IBM's SDLC definitions.

In the FLEXIBLE mode, reception or transmission of each frame by the SIU is performed under the control of the CPU. In this mode the 8044 can be either a primary station or a secondary station.

In both AUTO and FLEXIBLE modes, short frames, aborted frames, or frames which have had CRC's are ignored by the SIU.

The basic format of an SDLC frame is as follows:

| Flag | Address | Control | Information | FCS | Flag |

Format variations consist of omitting one or more of the fields in the SDLC frame. For example, a supervisory frame is formed by omitting the information field. Supervisory frames are used to confirm received frames, indicate ready or busy conditions, and to report errors. More details on frame formats are given in the SDLC Frame Format Options section, below.

18.4.1 AUTO Mode

To enable the SIU to receive a frame in AUTO mode, the 8044 CPU sets up a receive buffer. This is done by writing two registers—Receive Buffer Start (RBS) Address and Receive Buffer Length (RBL).

The SIU receives the frame, examines the control byte, and takes the appropriate action. If the frame is an information frame, the SIU will load the receive buffer, interrupt the CPU (to have the receive buffer read), and make the required acknowledgement to the primary station. Details on these processes are given in the Operation section, below.

In addition to receiving the information frames, the SIU in AUTO mode is capable of responding to the following commands (found in the control field of supervisory frames) from the primary station:

- RR (Receive Ready): Acknowledges that the Primary station has correctly received numbered frames up through \( N_R - 1 \), and that it is ready to receive frame \( N_R \).
- RNR (Receive Not Ready): Indicates a temporary busy condition (at the primary station) due to buffering or other internal constraints. The quantity \( N_R \) in the control field indicates the number of the frame expected after the busy condition ends, and may be used to acknowledge the correct reception of the frames up through \( N_R - 1 \).
- REJ (Reject): Acknowledges the correct reception of frames up through \( N_R - 1 \), and requests transmission or retransmission starting at frame \( N_R \). The 8044 is capable of retransmitting at most the previous frame, and then only if it is still available in the transmit buffer.
- UP (Unnumbered Poll): Also called NSP (Non-Sequenced Poll) or ORP (Optional Response Poll). This command is used in the loop configuration.

To enable the SIU to transmit an information frame in AUTO mode, the CPU sets up a transmit buffer. This is done by writing two registers—Transmit Buffer Start (TBS) Address and Transmit Buffer Length (TBL), and filling the transmit buffer with the information to be transmitted.

When the transmit buffer is full, the SIU can automatically (without CPU intervention) send an information frame (I-frame) with the appropriate sequence numbers, when the data link becomes available (when the 8044 is polled for information). After the SIU has transmitted the I-frame, it waits for acknowledgement from the receiving station. If the acknowledgement is negative, the SIU retransmits the frame. If the acknowledgement is positive, the SIU interrupts the CPU, to indicate that the transmit buffer may be reloaded with new information.
Figure 18-3. Serial Interface Timing—Clocked Mode
Figure 18-4. Serial Interface Timing—Self Clocked Mode

8044

RXD 10
TXD 11
16X/32X 15

DATA
GA
ONE'S
SHUT-OFF SEQ
DATA

RXD
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

TXD
0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1

DATA (BIT-DELAYED)
GA CHANGED TO FLAG
TRANSMIT FRAME
DATA (BIT-DELAYED)

TRANSMISSION ABORTED BY SHUT-OFF SEQUENCE

EXTRA "1" INSERTED
In addition to transmitting the information frames, the SIU in AUTO mode is capable of sending the following responses to the primary station:

RR (Receive Ready): Acknowledges that the 8044 has correctly received numbered frames up through \( N_R - 1 \), and that it is ready to receive frame \( N_R \).

RNR (Receive Not Ready): Indicates a temporary busy condition (at the 8044) due to buffering or other internal constraints. The quantity \( N_R \) in the control field indicates the number of the frame expected after the busy condition ends, and acknowledges the correct reception of the frames up through \( N_R - 1 \).

18.4.2 FLEXIBLE Mode

In the FLEXIBLE (or non-auto) mode, all reception and transmission is under the control of the CPU. The full SDLC and HDLC protocols can be implemented, as well as any bit-synchronous variants of these protocols.

FLEXIBLE mode provides more flexibility than AUTO mode, but it requires more CPU overhead, and much longer recognition and response times. This is especially true when the CPU is servicing an interrupt that has higher priority than the interrupts from the SIU.

In FLEXIBLE mode, when the SIU receives a frame, it interrupts the CPU. The CPU then reads the control byte from the Receive Control Byte (RCB) register. If the received frame is an information frame, the CPU also reads the information from the receive buffer, according to the values in the Receive Buffer Start (RBS) address register and the Received Field Length (RFL) register.

In FLEXIBLE mode, the 8044 can initiate transmissions without being polled, and thus it can act as the primary station. To initiate transmission or to generate a response, the CPU sets up and enables the SIU. The SIU then formats and transmits the desired frame. Upon completion of the transmission, without waiting for a positive acknowledgement from the receiving station, the SIU interrupts the CPU.

18.5 8044 FRAME FORMAT OPTIONS

As mentioned above, variations on the basic SDLC frame consist of omitting one or more of the fields. The choice of which fields to omit, as well as the selection of AUTO mode versus FLEXIBLE mode, is specified by the settings of the following three bits in the Serial Mode Register (SMD) and the Status/Control Register (STS):

- **SMD Bit 0: NFCS (No Frame Check Sequence)**
- **SMD Bit 1: NB (Non-Buffered Mode—No Control Field)**
- **STS Bit 1: AM (AUTO Mode or Addressed Mode)**

Figure 18-5 shows how these three bits control the frame format.

The following paragraphs discuss some properties of the standard SDLC format, and the significance of omitting some of the fields.

18.5.1 Standard SDLC Format

The standard SDLC format consists of an opening flag, an 8-bit address field, and 8-bit control field, an \( n \)-byte information field, a 16-bit Frame Check Sequence (FCS), and a closing flag. The FCS is based on the CCITT-CRC polynomial \( X^{16} + X^{12} + X^3 + 1 \). The address and control fields may not be extended. Within the 8044, the address field is held in the Station Address (STAD) register, and the control field is held in the Receive Control Byte (RCB) or Transmit Control Byte (TCB) register. The standard SDLC format may be used in either AUTO mode or FLEXIBLE mode.

18.5.2 No Control Field (Non-Buffered Mode)

When the control field is not present, the RCB and TCB registers are not used. The information field begins immediately after the address field, or, if the address field is also absent, immediately after the opening flag. The entire information field is stored in the 8044's on-chip RAM. If there is no control field, FLEXIBLE mode must be used. Control information may, of course, be present in the information field, and in this manner the No Control Field option may be used for implementing extended control fields.

18.5.3 No Control Field and No Address Field

The No Address Field option is available only in conjunction with the No Control Field option. The STAD, RCB, and TCB registers are not used. When both these fields are absent, the information field begins immediately after the opening flag. The entire information field is stored in on-chip RAM. FLEXIBLE mode must be used. Formats without an address field have the following applications:

- **Point-to-point data links (where no addressing is necessary)**
- **Monitoring line activity (receiving all messages regardless of the address field)**
- **Extended addressing**
### FRAME OPTION

<table>
<thead>
<tr>
<th>FRAME OPTION</th>
<th>NFCS</th>
<th>NB</th>
<th>AM</th>
<th>FRAME FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard SDLC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F A C I FCS F</td>
</tr>
<tr>
<td>FLEXIBLE Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard SDLC</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F A C I FCS F</td>
</tr>
<tr>
<td>AUTO Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Control Field</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F A I FCS F</td>
</tr>
<tr>
<td>FLEXIBLE Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Control Field</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F I FCS F</td>
</tr>
<tr>
<td>No Address Field</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLEXIBLE Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No FCS Field</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F A C I F</td>
</tr>
<tr>
<td>FLEXIBLE Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No FCS Field</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F A C I F</td>
</tr>
<tr>
<td>AUTO Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No FCS Field</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F A I F</td>
</tr>
<tr>
<td>No Control Field</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLEXIBLE Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No FCS Field</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F I F</td>
</tr>
<tr>
<td>No Control Field</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Address Field</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLEXIBLE Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key to Abbreviations:**

- F = Flag (01111110)
- A = Address Field
- C = Control Field
- I = Information Field
- FCS = Frame Check Sequence

**Note:** The AM bit is AUTO mode control bit when NB = 0, and Address Mode control bit when NB = 1.

---

**Figure 18-5. Frame Format Options**

#### 18.5.4 No FCS Field

In the normal case (NFCS=0), the last 16 bits before the closing flag are the Frame Check Sequence (FCS) field. These bits are not stored in the 8044's RAM. Rather, they are used to compute a cyclic redundancy check (CRC) on the data in the rest of the frame. A received frame with a CRC error (incorrect FCS) is ignored. In transmission, the FCS field is automatically computed by the SIU, and placed in the transmitted frame just prior to the closing flag.

The NFCS bit (SMD Bit 0) gives the user the capability of overriding this automatic feature. When this bit is set (NFCS=1), all bits from the beginning of the information field to the beginning of the closing flag are treated as part of the information field, and are stored in the on-chip RAM. No FCS checking is done on the received frames, and no FCS is generated for the transmitted frames. The No FCS Field option may be used in conjunction with any of the other options. It is typically used in FLEXIBLE mode, although it does not strictly include AUTO mode. Use of the No FCS Field option...
AUTO Mode may, however, result in SDLC protocol violations, since the data integrity is not checked by the SIU.

Formats without an FCS field have the following applications:
- Receiving and transmitting frames without verifying data integrity
- Using an alternate data verification algorithm
- Using an alternate CRC-16 polynomial (such as $X^{16} + X^5 + X^2 + 1$), or a 32-bit CRC
- Performing data link diagnosis by forcing false CRCs to test error detection mechanisms

In addition to the applications mentioned above, all of the format variations are useful in the support of non-standard bit-synchronous protocols.

### 18.6 HDLC

In addition to its support of SDLC communications, the 8044 also supports some of the capabilities of HDLC. The following remarks indicate the principal differences between SDLC and HDLC:

- **HDLC** permits any number of bits in the information field, whereas SDLC requires a byte structure (multiple of 8 bits). The 8044 itself operates on byte boundaries, and thus it restricts fields to multiples of 8 bits.
- **HDLC** provides functional extensions to SDLC: an unlimited address field is allowed, and extended frame number sequencing.
- **HDLC** does not support operation in loop configurations.

### 18.7 SIU SPECIAL FUNCTION REGISTERS

The 8044 CPU communicates with and controls the SIU through hardware registers. These registers are accessed using direct addressing. The SIU special function registers (SIU SFRs) are of three types:

- Control and Status Registers
- Parameter Registers
- ICE Support Registers

#### 18.7.1 Control and Status Registers

There are three SIU Control and Status Registers:
- **Serial Mode Register (SMD)**
- **Status/Command Register (STS)**
- **Send/Receive Count Register (NSNR)**

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below (see also the More Details on Registers section).

#### SMD: Serial Mode Register (byte-addressable)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>NFCS</td>
<td>No FCS field in the SDLC frame.</td>
</tr>
<tr>
<td>6</td>
<td>NB</td>
<td>Non-Buffered mode. No control field in the SDLC frame.</td>
</tr>
<tr>
<td>5</td>
<td>PFS</td>
<td>Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.</td>
</tr>
<tr>
<td>4</td>
<td>LOOP</td>
<td>Loop configuration.</td>
</tr>
<tr>
<td>3</td>
<td>NRZI</td>
<td>NRZI coding option.</td>
</tr>
<tr>
<td>2</td>
<td>SCM0</td>
<td>Select Clock Mode — Bit 0</td>
</tr>
<tr>
<td>1</td>
<td>SCM1</td>
<td>Select Clock Mode — Bit 1</td>
</tr>
<tr>
<td>0</td>
<td>SCM2</td>
<td>Select Clock Mode — Bit 2</td>
</tr>
</tbody>
</table>

The individual bits of the Serial Mode Register are as follows:

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>NFCS</td>
<td>No FCS field in the SDLC frame.</td>
</tr>
<tr>
<td>6</td>
<td>NB</td>
<td>Non-Buffered mode. No control field in the SDLC frame.</td>
</tr>
<tr>
<td>5</td>
<td>PFS</td>
<td>Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.</td>
</tr>
<tr>
<td>4</td>
<td>LOOP</td>
<td>Loop configuration.</td>
</tr>
<tr>
<td>3</td>
<td>NRZI</td>
<td>NRZI coding option.</td>
</tr>
<tr>
<td>2</td>
<td>SCM0</td>
<td>Select Clock Mode — Bit 0</td>
</tr>
<tr>
<td>1</td>
<td>SCM1</td>
<td>Select Clock Mode — Bit 1</td>
</tr>
<tr>
<td>0</td>
<td>SCM2</td>
<td>Select Clock Mode — Bit 2</td>
</tr>
</tbody>
</table>

The SCM bits decode as follows:

<table>
<thead>
<tr>
<th>SCM</th>
<th>Clock Mode</th>
<th>Data Rate (Bits/sec)*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>Externally clocked</td>
<td>0–2.4M**</td>
</tr>
<tr>
<td>0 0 1</td>
<td>Undefined</td>
<td>0.8–32.0K</td>
</tr>
<tr>
<td>0 1 0</td>
<td>Self clocked, timer overflow</td>
<td>244–62.5K</td>
</tr>
<tr>
<td>0 1 1</td>
<td>Undefined</td>
<td>0.8–32.0K</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Self clocked, external 16x</td>
<td>0–375K</td>
</tr>
</tbody>
</table>
SCM

Clock Mode Data Rate

<table>
<thead>
<tr>
<th>SCM</th>
<th>Clock Mode</th>
<th>Data Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>012</td>
<td>Self clocked, external 32x</td>
<td>0-187.5k</td>
</tr>
<tr>
<td>101</td>
<td>Self clocked, internal fixed</td>
<td>375k</td>
</tr>
<tr>
<td>111</td>
<td>Self clocked, internal fixed</td>
<td>187.5k</td>
</tr>
</tbody>
</table>

*Based on a 12 Mhz crystal frequency

**0-1M bps in loop configuration

STS: Status/Command Register (bit-addressable)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>TBF</td>
<td>Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.</td>
</tr>
<tr>
<td>6</td>
<td>RBE</td>
<td>Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.</td>
</tr>
<tr>
<td>5</td>
<td>RTS</td>
<td>Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.</td>
</tr>
<tr>
<td>4</td>
<td>SI</td>
<td>SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.</td>
</tr>
<tr>
<td>3</td>
<td>BOV</td>
<td>Receive Buffer Overrun. BOV may be set or cleared by the SIU.</td>
</tr>
<tr>
<td>2</td>
<td>OPB</td>
<td>Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.</td>
</tr>
<tr>
<td>1</td>
<td>AM</td>
<td>AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.</td>
</tr>
<tr>
<td>0</td>
<td>RBP</td>
<td>Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.</td>
</tr>
</tbody>
</table>

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU’s access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles (‘JBC/B, REL’, and ‘MOV/B,C’) should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit # Name Description

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STS.0</td>
<td>RBP</td>
<td>Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.</td>
</tr>
<tr>
<td>STS.1</td>
<td>AM</td>
<td>AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.</td>
</tr>
<tr>
<td>STS.2</td>
<td>OPB</td>
<td>Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.</td>
</tr>
<tr>
<td>STS.3</td>
<td>BOV</td>
<td>Receive Buffer Overrun. BOV may be set or cleared by the SIU.</td>
</tr>
<tr>
<td>STS.4</td>
<td>SI</td>
<td>SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.</td>
</tr>
<tr>
<td>STS.5</td>
<td>RTS</td>
<td>Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.</td>
</tr>
</tbody>
</table>

NSNR: Send/Receive Count Register (bit-addressable)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSNR.0</td>
<td>SER</td>
<td>Receive Sequence Error: NS (P) ≠ NR (S)</td>
</tr>
<tr>
<td>NSNR.1</td>
<td>NR0</td>
<td>Receive Sequence Counter—Bit 0</td>
</tr>
<tr>
<td>NSNR.2</td>
<td>NR1</td>
<td>Receive Sequence Counter—Bit 1</td>
</tr>
<tr>
<td>NSNR.3</td>
<td>NR2</td>
<td>Receive Sequence Counter—Bit 2</td>
</tr>
<tr>
<td>NSNR.4</td>
<td>SES</td>
<td>Send Sequence Error: NR (P) ≠ NS (S) and NR (P) ≠ NS (S) + 1</td>
</tr>
<tr>
<td>NSNR.5</td>
<td>NS0</td>
<td>Send Sequence Counter — Bit 0</td>
</tr>
<tr>
<td>NSNR.6</td>
<td>NS1</td>
<td>Send Sequence Counter — Bit 1</td>
</tr>
<tr>
<td>NSNR.7</td>
<td>NS2</td>
<td>Send Sequence Counter — Bit 2</td>
</tr>
</tbody>
</table>

18.7.2 Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.
The eight parameter registers are as follows:

**STAD: Station Address Register (byte-addressable)**

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS = 0 and RBE = 0). Normally, STAD is accessed only during initialization.

**TBS: Transmit Buffer Start Address Register (byte-addressable)**

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF = 0).

**TBL: Transmit Buffer Length Register (byte-addressable)**

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL = 0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF = 0).

**NOTE:** The transmit and receive buffers are not allowed to “wrap around” in the on-chip RAM. A “buffer end” is automatically generated if address 191 (BFH) is reached.

**TCB: Transmit Control Byte Register (byte-addressable)**

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF = 0). The Ns and Nc counters are not used in the NON-AUTO mode.

**RBS: Receive Buffer Start Address Register (byte-addressable)**

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE = 0).

**RBL: Receive Buffer Length Register (byte-addressable)**

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip RAM allocated for the received I-field. RBL = 0 is valid. The CPU should write RBL only when RBE = 0.

**RFL: Receive Field Length Register (byte-addressable)**

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL = 0 is valid. RFL should be accessed by the CPU only when RBE = 0.

**RCB: Receive Control Byte Register (byte-addressable)**

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE = 0.

**18.7.3 ICE Support Registers**

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can exercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

Among the SIU SFRs are the following registers that support the operation of the ICE:

**DMA CNT: DMA Count Register (byte-addressable)**

The DMA Count register (Address CFH) indicates the number of bytes remaining in the information block that is currently being used.

**FIFO: Three-Byte (byte-addressable)**

The Three-Byte FIFO (Address DDH, DEH, and DFH) is used between the eight-bit shift register and the information buffer when an information block is received.
SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register.

The SIUST register can serve as a helpful aid to determine which field of a receive frame that the SIU expects next. The table below will help in debugging 8044 reception problems.

<table>
<thead>
<tr>
<th>SIUST VALUE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>01H</td>
<td>Waiting for opening flag.</td>
</tr>
<tr>
<td>08H</td>
<td>Waiting for address field.</td>
</tr>
<tr>
<td>10H</td>
<td>Waiting for control field.</td>
</tr>
<tr>
<td>18H</td>
<td>Waiting for first byte of I field. This state is only entered if a FCS is expected. It pushes the received byte onto the top of the FIFO.</td>
</tr>
<tr>
<td>20H</td>
<td>Waiting for second byte of I field. This state always follows state 18H</td>
</tr>
<tr>
<td>28H</td>
<td>Waiting for I field byte. This state can be entered from state 20H or from states 01H, 08H, or 10H depending upon the SIU's mode configuration. (Each time a byte is received, it is pushed onto the top of the FIFO and the byte at the bottom is put into memory. For no FCS formatted frames, the FIFO is collapsed into a single register).</td>
</tr>
<tr>
<td>30H</td>
<td>Waiting for the closing flag after having overflowed the receive buffer. Note that even if the receive frame overflows the assigned receive buffer length, the FCS is still checked.</td>
</tr>
</tbody>
</table>

Examples of SIUST status sequences for different frame formats are shown below. Note that status changes after acceptance of the received field byte.

### 18.8 OPERATION

The SIU is initialized by a reset signal (on pin 9), followed by write operations to the SIU SFRs. Once initialized, the SIU can function in AUTO mode or NON-AUTO mode. Details are given below.

#### Table 18-1. SIUST Status Sequences

<table>
<thead>
<tr>
<th>Example 1:</th>
<th>Frame Format</th>
<th>SIUST Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Format</td>
<td>(Idle) F A C I</td>
<td>01 01 08 10 18 20 28 28 F0</td>
</tr>
<tr>
<td>SIUST Value</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame Option</th>
<th>NFCS</th>
<th>NB</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 2:</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Frame Format</td>
<td>(Idle) F A I</td>
<td>01 01 08 18 20 28 28 01</td>
<td></td>
</tr>
<tr>
<td>SIUST Value</td>
<td></td>
<td>FCS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example 3:</th>
<th>Frame Format</th>
<th>SIUST Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Format</td>
<td>(Idle) F I</td>
<td>01 01 18 20 28 28 01</td>
</tr>
<tr>
<td>SIUST Value</td>
<td></td>
<td>FCS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame Option</th>
<th>NFCS</th>
<th>NB</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 4:</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Frame Format</td>
<td>(Idle) F A I F</td>
<td>01 01 08 28 01</td>
<td></td>
</tr>
<tr>
<td>SIUST Value</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example 5:</th>
<th>Frame Format</th>
<th>SIUST Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame Format</td>
<td>(Idle) F I F</td>
<td>01 01 28 01</td>
</tr>
<tr>
<td>SIUST Value</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame Option</th>
<th>NFCS</th>
<th>NB</th>
<th>AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 6:</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Frame Format</td>
<td>(Idle) F I 1 OVERFLOW FCS F</td>
<td>01 01 18 20 28 30 30 01</td>
<td></td>
</tr>
<tr>
<td>SIUST Value</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

18-12
18.8.1 Initialization

Figure 18-6 is the SIU. Registers SMD, STS, and NSNR are cleared by reset. This puts the 8044 into an idle state—neither receiving nor transmitting. The following registers must be initialized before the 8044 leaves the idle state:

- STAD—to establish the 8044's SDLC station address.
- SMD—to configure the 8044 for the proper operating mode.
- RBS, RBL—to define the area in RAM allocated for the Receive Buffer.

TBS, TBL—to define the area in RAM allocated for the Transmit Buffer.

Once these registers have been initialized, the user may write to the STS register to enable the SIU to leave the idle state, and to begin transmits and/or receives.

Setting RBE to 1 enables the SIU for receive. When RBE = 1, the SIU monitors the received data stream for a flag pattern. When a flag pattern is found, the SIU enters Receive mode and receives the frame.

Setting RTS to 1 enables the SIU for transmit. When RTS = 1, the SIU monitors the received data stream for a GA pattern (loop configuration) or waits for a CTS

Figure 18-6. SIU State Diagram
(non-loop configuration). When the GA or CTS arrives, the SIU enters Transmit mode and transmits a frame.

In AUTO mode, the SIU sets RTS to enable automatic transmissions of appropriate responses.

### 18.8.2 AUTO Mode

Figure 18-7 illustrates the receive operations in AUTO mode. The overall operation is shown in Figure 18-7a. Particular cases are illustrated in Figures 18-7b through 18-7j. If any Unnumbered Command other than UP is received, the AM bit is cleared and the SIU responds as if in the FLEXIBLE mode, by interrupting the CPU for supervision. This will also happen if a BOV or SES condition occurs. If the received frame contains a poll, the SIU sets the RTS bit to generate a response.

Figure 18-8 illustrates the transmit operations in AUTO mode. When the SIU gets the opportunity to transmit, and if the transmit buffer is full, it sends an I-frame. Otherwise, it sends an RR if the buffer is free, or an RNR if the buffer is protected. The sequence counters NS and NR are used to construct the appropriate control fields.

Figure 18-9 shows how the CPU responds to an SI (serial interrupt) in AUTO mode. The CPU tests the AM bit (in the STS register). If AM = 1, it indicates that the SIU has received either an I-frame, or a positive response to a previously transmitted I-frame.

### 18.8.3 FLEXIBLE Mode

Figure 18-10 illustrates the receive operations in NON-AUTO mode. When the SIU successfully completes a task, it clears RBF and interrupts the CPU by setting SI to 1. The exact CPU response to SI is determined by software. A typical response is shown in Figure 18-11.

Figure 18-12 illustrates the transmit operations in FLEXIBLE mode. The SIU does not wait for a positive acknowledge response to the transmitted frame. Rather, it interrupts the CPU (by setting SI to 1) as soon as it finishes transmitting the frame. The exact CPU response to SI is determined by software. A typical response is shown in Figure 18-13. This response results in another transmit frame being set up. The sequence of operations shown in Figure 18-13 can also be initiated by the CPU, without an SI. Thus the CPU can initiate a transmission in FLEXIBLE mode without a poll, simply by setting the RTS bit in the STS register. The RTS bit is always used to initiate a transmission, but it is applied to the RTS pin only when a non-loop configuration is used.

### 18.8.4 8044 Data Link Particulars

The following facts should be noted:

1) In a non-loop configuration, one or two bits are transmitted before the opening flag. This is necessary for NRZI synchronization.

2) In a non-loop configuration, one to eight extra dribble bits are transmitted after the closing flag. These bits are a zero followed by ones.

3) In a loop configuration, when a GA is received and the 8044 begins transmitting, the sequence is 0111110101111110... (FLAG, 1, FLAG, ADDRESS, etc.). The first flag is created from the GA. The second flag begins the message.

4) CTS is sampled after the rising edge of the serial data, at about the center of the bit cell, except during a non-loop, externally clocked mode transmit, in which case it is sampled just after the falling edge.

5) The SIU does not check for illegal I-fields. In particular, if a supervisory command is received in AUTO mode, and if there is also an I-field, it will be loaded into the receive buffer (if RBP=0), but it cannot cause a BOV.

6) In relation to the Receive Buffer Protect facility, the user should set RFL to 0 when clearing RBP, such that, if the SIU is in the process of receiving a frame, RFL will indicate the proper value when reception of the frame has been completed.

### 18.8.5 Turn Around Timing

In AUTO mode, the SIU generates an RTS immediately upon being polled. Assuming that the 8044 sends an information frame in response to the poll, the primary station sends back an acknowledgement. If, in this acknowledgement, the 8044 is polled again, a response may be generated even before the CPU gets around to processing the interrupt caused by the acknowledgement. In such a case, the response would be an RR (or RNR), since TBF would have been set to 0 by the SIU, due to the acknowledge.

If the system designer does not wish to take up channel time with RR responses, but prefers to generate a new I-frame as a response, there are several ways to accomplish this:

1) Operate the 8044 in FLEXIBLE mode.

2) Specify that the master should never acknowledge and poll in one message. This is typically how a loop system operates, with the poll operation confined to the UP command. This leaves plenty of time for the
Figure 18-7a. SIU AUTO Mode Receive Flowchart—General
Figure 18-7b. SIU AUTO Mode Receive Flowchart—Unknown Command
Figure 18-7c. SIU AUTO Mode Receive Flowchart—Unnumbered Poll
Figure 18-7d. SIU AUTO Mode Receive Flowchart—Supervisory Command
Figure 18-7e. SIU AUTO Mode Receive Flowchart—1 Command: Prior Transmitted I-Field Confirmed, Current Received I-Field in Sequence
Figure 18-7f. SIU AUTO Mode Receive Flowchart—1 Command: Prior Transmitted I-Field Not Confirmed, Current Received I-Field in Sequence
Figure 18-7g. SIU AUTO Mode Receive Flowchart—Command: Sequence Error Send, Current Received I-Field in Sequence
Figure 18-7h. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed Sequence Error Receive
Figure 18-7f. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Sequence Error Receive
Figure 18-7j. SIU SUTO Mode Receive Flowchart—1 Command: Sequence Error Send and Sequence Error Receive
Figure 18-8. SIU AUTO Mode Transmit Flowchart
"SI" SET
[SIU]
INTERUPT]

CLEAR "SI"

TEST "AM"

UNKNOWN
COMMAND OR
TRANSMIT SEQ
ERR OR BOV

= 0

= 1

TEST "TBF"

TRANS BUFFER EMPTY

DOES CPU HAVE DATA?
NO
YES

LOAD I-FIELD INTO XMIT BUFFER

SET "TBF"

TRANSMIT BUFFER FULL

RETURN

DOES CPU HAVE DATA?
NO
YES

PROCESS INFORMATION OR SET "RBP"

SET "RBE"

Figure 18-9. AUTO Mode Response to "SI"
Figure 18-10. SIU FLEXIBLE Mode Receive Flowchart
Figure 18-11. FLEXIBLE Mode Response to Receive "SI"
Figure 18-12. SIU FLEXIBLE Mode Transmit Flowchart
Figure 18-13. FLEXIBLE Mode Response to Transmit "SI"
8044 to get its transmit buffer loaded with new information after an acknowledge.

3) The 8044 CPU can clear RTS. This will prevent a response from being sent, or abort it if it is already in progress. A system using external RTS/CTS handshaking could use a one-shot to delay RTS or CTS, thereby giving the CPU more time to disable the response.

18.9 MORE DETAILS ON SIU HARDWARE

The SIU divides functionally into two sections—a bit processor (BIP) and a byte processor (BYP)—sharing some common timing and control logic. As shown in Figure 18-14, the BIP operates between the serial port pins and the SIU bus, and performs all functions necessary to transmit/receive a byte of data to/from the serial data stream. These operations include shifting, NRZI encoding/decoding, zero insertion/deletion, and FCS generation/checking. The BIP manipulates bytes of data to perform message formatting, and other transmitting and receiving functions. It operates between the SIU bus (SIB) and the 8044's internal bus (IB).

The interface between the SIU and the CPU involves an interrupt and some locations in on-chip RAM space which are managed by the BYP.

The maximum possible data rate for the serial port is limited to 1/2 the internal clock rate. This limit is imposed by both the maximum rate of DMA to the on-chip RAM, and by the requirements of synchronizing to an external clock. The internal clock rate for an 8044 running on a 12 MHz crystal is 6 MHz. Thus the maximum 8044 serial data rate is 3 MHz. This data rate drops down to 2.4 MHz when time is allowed for external clock synchronization.

18.9.1 The Bit Processor

In the asynchronous (self clocked) modes the clock is extracted from the data stream using the on-chip digital phase-locked-loop (DPLL). The DPLL requires a clock input at 16 times the data rate. This 16 X clock may originate from SCLK, Timer 1 Overflow, or PH2 (one half the oscillator frequency). The extra divide by-two described above allows these sources to be treated alternatively as 32 X clocks.

The DPLL is a free-running four-bit counter running off the 16 X clock. When a transition is detected in the receive data stream, a count is dropped (by suppressing the carry-in) if the current count value is greater than 8. A count is added (by injecting a carry into the second stage rather than the first) if the count is less than 8. No adjustment is made if the transition occurs at the count of 8. In this manner the counter locks in on the point at which transitions in the data stream occur at the count of 8, and a clock pulse is generated when the counter overflows to 0.

In order to perform NRZI decoding, the NRZI decoder compares each bit of input data to the previous bit. There are no clock delays in going through the NRZI decoder.

The zero insert/delete circuitry (ZID) performs zero insertion/deletion, and also detects flags, GA's (Go-Ahead's), and aborts (same as GA's) in the data stream. The pattern 1111110 is detected as an early GA, so that the GA may be turned into a flag for loop mode transmission.

The shut-off detector monitors the receive data stream for a sequence of eight zeros, which is a shut-off command for loop mode transmissions. The shut-off detector is a three-bit counter which is cleared whenever a one is found in the receive data stream. Note that the ZID logic could not be used for this purpose, because the receive data must be monitored even when the ZID is being used for transmission.

As an example of the operation of the bit processor, the following sequence occurs in relation to the receive data:

1) RXD is sampled by SCLK, and then synchronized to the internal processor clock (IPC).

2) If the NRZI mode is selected, the incoming data is NRZI decoded.

3) When receiving other than the flag pattern, the ZID deletes the '0' after 5 consecutive '1's (during transmission this zero is inserted). The ZID locates the byte boundary for the rest of the circuitry. The ZID deletes the '0's by preventing the SR (shift register) from receiving a clocking pulse.

4) The FCS (which is a function of the data between the flags—not including the flags) is initialized and started at the detection of the byte boundary at the end of the opening flag. The FCS is computed each bit boundary until the closing flag is detected. Note that the received FCS has gone through the ZID during transmission.

18.9.2 The Byte Processor

Figure 18-15 is a block diagram of the byte processor (BYP). The BYP contains the registers and controllers necessary to perform the data manipulations associated with SDLC communications. The BYP registers may be read or written by the CPU over the 8044's internal bus.
(IB), using standard 8044 hardware register operations. The 8044 register select PLA controls these operations. Three of the BYP registers connect to the IB through the IBS, a sub-bus which also connects to the CPU interrupt control registers.

Simultaneous access of a register by both the IB and the SIB is prevented by timing. In particular, RAM access is restricted to alternate internal processor cycles for the CPU and the SIU, in such a way that collisions do not occur.

As an example of the operation of the byte processor, the following sequence occurs in relation to the receive data:

1) Assuming that there is an address field in the frame, the BYP takes the station address from the register file into temporary storage. After the opening flag, the next field (the address field) is compared to the station address in the temporary storage. If a match occurs, the operation continues.

2) Assuming that there is a control field in the frame, the BYP takes the next byte and loads it into the RCB register. The RCB register has the logic to update the NSNR register (increment receive count, set SES and SER flags, etc.).

3) Assuming that there is an information field, the next byte is dumped into RAM at the RBS location. The DMA CNT (RBL at the opening flag) is loaded from the DMA CNT register into the RB register and decremented. The RFL is then loaded into the RB register, incremented, and stored back into the register file.
4) This process continues until the DMA CNT reaches zero, or until a closing flag is received. Upon either event, the BYP updates the status, and, if the CRC is good, the NSNR register.

18.10 DIAGNOSTICS

An SIU test mode has been provided, so that the on-chip CPU can perform limited diagnostics on the SIU. The test mode utilizes the output latches for P3.0 and P3.1 (pins 10 and 11). These port 3 pins are not useful as output ports, since the pins are taken up by the serial port functions. Figure 18-16 shows the signal routing associated with the SIU test mode.

Writing a 0 to P3.1 enables the serial test mode (P3.1 is set to 1 by reset). In test mode the P3.0 bit is mapped into the received data stream, and the 'write port 3' control signal is mapped into the SCLK path in place of T1. Thus, in test mode, the CPU can send a serial data stream to the SIU by writing to P3.0. The transmit data stream can be monitored by reading P3.1. Each successive bit is transmitted from the SIU by writing to any bit in Port 3, which generates SCLK.

In test mode, the P3.0 and P3.1 pins are placed in a high voltage, high impedance state. When the CPU reads P3.0 and P3.1 the logic level applied to the pin will be returned. In the test mode, when the CPU reads 3.1, the transmit data value will be returned, not the voltage on the pin. The transmit data remains constant for a bit time. Writing to P3.0 will result in the signal being outputted for a short period of time. However, since the signal is not latched, P3.0 will quickly return to a high voltage, high impedance state.
Figure 18-16: SIU Test Mode

- PIN 15 SCLK/ T1/ P35
- P35 OUTPUT LATCH
- CPU BUS
- TIMER 1 OVF
- SYS CLK
- SIU SERIAL DATA CLOCK
- SIU RECEIVE DATA STREAM
- SIU Transmit Data Stream
- PIN 19 I/O/ RXD/ P38
- P38 OUTPUT LATCH
- LOOP
- SELF TEST
- P31 OUTPUT LATCH
- READ PORT 3
- WRITE PORT 3
- SIU RECEIVE DATA STREAM
- SIU Transmit Data Stream
- PIN 11 DATA/ TXD/ P31
The serial test mode is disabled by writing a 1 to P3.1. Care must be taken that a 0 is never written to P3.1 in the course of normal operation, since this causes the test mode to be entered.

Figure 18-17 is an example of a simple program segment that can be imbedded into the user's diagnostic program. That example shows how to put the 8044 into "Loop-back mode" to test the basic transmitting and receiving functions of the SIU.

Loop-back mode is functionally equivalent to a hardwire connection between pins 10 and 11 on the 8044.

In this example, the 8044 CPU plays the role of the primary station. The SIU is in the AUTO mode. The CPU sends the SIU a supervisory frame with the poll bit set and an RNR command. The SIU responds with a supervisory frame with the poll bit set and an RR command.

The operation proceeds as follows:

Interrupts are disabled, and the self test mode is enabled by writing a zero to P3.1. This establishes P3.0 as the data path from the CPU to the SIU. CTS (clear-to-send) is enabled by writing a zero to P1.7. The station address is initialized by writing 08AH into the STAD (station address register).

The SIU is configured for receive operation in the clocked mode and in AUTO mode. The CPU then transmits a supervisory frame. This frame consists of an opening flag, followed by the station address, a control field indicating that this is a supervisory frame with an RNR command, and then a closing flag.

Each byte of the frame is transmitted by writing that byte into the A register and then calling the subroutine XMIT8. Two additional SCLKs are generated to guarantee that the last bits in the frame have been clocked into the SIU. Finally the CPU reads the status register (STS). If the operation has proceeded correctly, the status will be 072H. If it is not, the program jumps to the ERROR loop and terminates.

The SIU generates an SI (SIU interrupt) to indicate that it has received a frame. The CPU clears this interrupt, and then begins to monitor the data stream that is being generated by the SIU in response to what it has received. As each bit arrives (via P3.1), it is moved into the accumulator, and the CPU compares the byte in the accumulator with 07EH, which is the opening flag. When a match occurs, the CPU identifies this as byte boundary, and thereafter processes the information byte-by-byte.

The CPU calls the RCV8 subroutine to get each byte into the accumulator. The CPU performs compare operations on (successively) the station address, the control field (which contains the RR response), and the closing flag. If any of these do not compare, the program jumps to the ERROR loop. If no error is found, the program jumps to the DONE loop.
Figure 18-17. Loop-Back Mode Software
Application Examples
CHAPTER 19
8044 APPLICATION EXAMPLES

19.0 8044 APPLICATIONS EXAMPLES

19.1 INTERFACING THE 8044 TO A MICROPROCESSOR

The 8044 is designed to serve as an intelligent controller for remote peripherals. However, it can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capable of extensively off-loading link control functions for the CPU. In some applications, the 8044 can be used for communications preprocessing, in addition to data link control.

This section describes a sample hardware interface for attaching the 8044 to an 8088. It is general enough to be extended to other microprocessors such as the 8086 or the 80186.

OVERVIEW

A sample interface is shown in Figure 19-1. Transmission occurs when the 8088 loads a 64 byte block of memory with some known data. The 8088 then enables the 8237A to DMA this data to the 8044. When the 8044 has received all of the data from the 8237A, it sends the data in a SDLC frame. The frame is captured by the Spectron Datascope which displays it on a CRT in hex format.

In reception, the Datascope sends an SDLC information frame to the 8044. The 8044 receives the SDLC frame, buffers it, and sends it to the 8088's memory. In this example the 8044 is being operated in the NON-AUTO mode; therefore, it does not need to be polled by a primary station in order to transmit.

THE INTERFACE

The 8044 does not have a parallel slave port. The 8044's 32 I/O lines can be configured as a local microprocessor bus master. In this configuration, the 8044 can expand the ROM and RAM memory, control peripherals, and communicate with a microprocessor.

The 8044, like the 8051, does not have a Ready line, so there is no way to put the 8044 in wait state. The clock on the 8044 cannot be stopped. Dual port RAM could still be used, however, software arbitration would be the only way to prevent collisions. Another way to interface the 8044 with another CPU is to use a FIFO or queue between the two processors, and this was the method chosen for this design.

Figure 19-2 shows the schematic of the 8044/8088 interface. It involves two 8 bit tri-state latches, two SR flip-flops, and some logic gates (6 TTL packs). The circuitry implements a one byte FIFO. RS422 transceivers are used, which can be connected to a multidrop link. Figure 19-3 shows the 8088 and support circuitry; the memory and decoders are not shown. It is a basic 8088 Min Mode system with an 8237A DMA controller and an 8259A interrupt controller.

DMA Channel One transfers a block of memory to the tri-state latch, while Channel Zero transfers a block of data from the latch to 8088's memory. The 8044's Interrupt 0 signal vectors the CPU into a routine which reads from the internal RAM and writes to the latch. The 8044's Interrupt 1 signal causes the chip to read from the latch and write to its on-chip data RAM. Both DMA requests and acknowledges are active low.

Initially, when the power is applied, a reset pulse coming from the 8284A initializes the SR flip-flops. In this initialization state, the 8044's transmit interrupt and the 8088's transmit DMA request are active; however, the software keeps these signals disabled until either of the two processors are ready to transmit. The software leaves the receive signals enabled, unless the receive buffers are full. In this way either the 8088 or the 8044 are always ready to receive, but they must enable the transmit signal when they have prepared a block to transmit. After a block has been transmitted or received, the DMA and interrupt signals return to the initial state.

The receive and transmit buffer sizes for the blocks of data sent between the 8044 and the 8088 have a maximum fixed length. In this case the buffer size was 64 bytes. The buffer size must be less than 192 bytes to enable 8044 to buffer the data in its on-chip RAM. This design allows blocks of data that are less than 64 bytes, and accommodates networks that allow frames of varying size. The first byte transferred between the 8088 and the 8044 is the byte count to follow; thus the 8044 knows how many bytes to receive before it transmits the SDLC frame. However, when the 8044 sends data to the 8088's memory, the 8237A will not know if the 8044 will send less than the count the 8237A was programmed for. To solve this problem, the 8237A is operated in the single mode. The 8044 uses an I/O bit to generate an interrupt request to the 8259A. In the 8088's interrupt routine, the 8237A's receive DMA channel is disabled, thus allowing blocks of data less than 64 bytes to be received.

THE SOFTWARE

The software for the 8044 and the 8088 is shown in Table 19-1. The 8088 software was written in PL/M86, and the 8044 software was written in assembly language.

The 8044 software begins by initializing the stack, interrupt priorities, and triggering types for the interrupts. At this point, the SIU parameter registers are initialized. The receive and transmit buffer starting addresses and lengths are loaded for the on-chip DMA. This DMA is for the serial port. The serial station address and the transmit control bytes are loaded too.

*Datascope is a trademark of Spectron Inc.
Once the initialization has taken place, the SIU interrupt is enabled, and the external interrupt which receives bytes from the 8088 is enabled. Setting the 8044's Receive Buffer Empty (RBE) bit enables the receiver. If this bit is reset, no serial data can be received. The 8044 then waits in a loop for either RECEIVE DMA interrupt or the SERIAL INT interrupt.

The RECEIVE DMA interrupt occurs when the 8237A is transferring a block of data to the 8044. The first time this interrupt occurs, the 8044 reads the latch and loads the count value into the R2 register. On subsequent interrupts, the 8044 reads the latch, loads the data into the transmit buffer, and decrements R2. When R2 reaches zero, the interrupt routine sends the data in an SDLC frame, and disables the RECEIVE DMA interrupt. After the frame has been transmitted, a serial interrupt is generated. The SERIAL INT routine detects that a frame has been transmitted and re-enables the RECEIVE DMA interrupt. Thus, while the frame is being transmitted through the SIU, the 8237A is inhibited from sending data to the 8044's transmit buffer.

The TRANSMIT DMA routine sends a block of data from the 8044's receive buffer to the 8088's memory. Normally this interrupt remains disabled. However, if a serial interrupt occurs, and the SERIAL INT routine detects that a frame has been received, it calls the SEND subroutine. The SEND subroutine loads the number of bytes which were received in the frame into the receive buffer. Register R1 points to the receive buffer and R2 is loaded with the count. The TRANSMIT DMA interrupt is enabled, and immediately upon returning from the SERIAL INT routine, the interrupt is acknowledged. Each time the TRANSMIT DMA interrupt occurs, a byte is read from the receive buffer, written to the latch, and R2 is decremented. When R2 reaches 0, the TRANSMIT DMA interrupt is disabled, the SIU receiver is re-enabled, and the 8044 interrupts the 8088.

The 8088 software simply transmits a block of data and receives a block of data, then stops. The software begins by initializing the 8237A, and the 8259A. It then loads a block of memory with some data and enables the 8237A to transmit the data. In the meantime the 8088 waits in a loop. After a block of data is received from the 8044, the 8088 is interrupted, and it shuts off the 8237A receive DMA.

CONCLUSION

For the software shown in Table 19-1, the transfer rate from the 8088's memory to the 8044 was measured at 75K bytes/sec. This transfer rate largely depends upon the number of instructions in the 8044's interrupt service routine. Fewer instructions result in a higher transfer rate.

There are many ways of interfacing the 8044 locally to another microprocessor: FIFO's, dual port RAM with software arbitration, and 8255's are just a few. Alternative approaches, which may be more optimal for certain applications, are certainly possible.
Table 19-1. Transmit and Receive Software for an 8044/8088 System

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$debug</td>
<td>title</td>
<td>(8044/8088 INTERFACE)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0000</td>
<td>FIRST_BYTE</td>
<td>BIT 0 ; FLAG</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0000</td>
<td>ORG</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0000 8024</td>
<td>SJMP</td>
<td>INIT</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0026</td>
<td>ORG</td>
<td>26H</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0026 7581AA</td>
<td>INIT:</td>
<td>MOV SP, #170 ; INITIALIZE STACK</td>
</tr>
<tr>
<td>12</td>
<td>0029 75B800</td>
<td>MOV</td>
<td>IP, #00 ; ALL INTERRUPTS ARE EQUAL PRIORITY</td>
</tr>
<tr>
<td>13</td>
<td>002C 75C954</td>
<td>MOV</td>
<td>SMD, #54H ; TIMER 1 OVERFLOW, NRZI, PRE-FRAME SYNC</td>
</tr>
<tr>
<td>14</td>
<td>002F 758844</td>
<td>MOV</td>
<td>TCON, #44H ; EDGE TRIGGERED EXTERNAL INTERRUPT 1</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0032 758DEC</td>
<td>MOV</td>
<td>TH1, #0ECH ; INITIALIZE TIMER, 3125 BPS</td>
</tr>
<tr>
<td>17</td>
<td>0035 758920</td>
<td>MOV</td>
<td>TMOD, #20H ; TIMER 1 AUTO RELOAD</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0038 75DC6A</td>
<td>MOV</td>
<td>TBS, #106 ; SET UP SIU PARAMETER REGISTERS</td>
</tr>
<tr>
<td>20</td>
<td>003B 75DB40</td>
<td>MOV</td>
<td>TBL, #64</td>
</tr>
<tr>
<td>21</td>
<td>003C 75CC2A</td>
<td>MOV</td>
<td>RBS, #64</td>
</tr>
<tr>
<td>22</td>
<td>0041 75CB40</td>
<td>MOV</td>
<td>RBL, #64</td>
</tr>
<tr>
<td>23</td>
<td>0044 75CE55</td>
<td>MOV</td>
<td>STAD, #55H</td>
</tr>
<tr>
<td>24</td>
<td>0047 75DA11</td>
<td>MOV</td>
<td>TCB, #00010001B ; RR, P/F = 1</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>004A 901000</td>
<td>MOV</td>
<td>DPTR, #100H ; DPTR POINTS TO TRI-STATE LATCH</td>
</tr>
<tr>
<td>27</td>
<td>004D D200</td>
<td>SETB</td>
<td>FIRST_BYTE ; FLAG TO INDICATE FIRST BYTE</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>FOR RECEIVE INTERRUPT ROUTINE</td>
</tr>
<tr>
<td>29</td>
<td>004F D2CE</td>
<td>SETB</td>
<td>RBE ; READY TO RECEIVE</td>
</tr>
<tr>
<td>30</td>
<td>0051 75A894</td>
<td>MOV</td>
<td>IE, #1001000B ; ENABLE RECEIVE DMA AND SIU INTERRUPT</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0054 80FE</td>
<td>SJMP</td>
<td>$ ; WAIT HERE FOR INTERRUPTS</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>0056 80FE</td>
<td>ERROR:</td>
<td>SJMP ERROR</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td>+1 SEJ</td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>0058 85CD29</td>
<td>SEND:</td>
<td>MOV 41, RFL ; FIRST BYTE IN BLOCK IS COUNT</td>
</tr>
<tr>
<td>40</td>
<td>005B 7929</td>
<td>MOV</td>
<td>R1, #41 ; POINT TO BLOCK OF DATA</td>
</tr>
<tr>
<td>41</td>
<td>005D AACD</td>
<td>MOV</td>
<td>R2, RFL ; LOAD COUNT</td>
</tr>
<tr>
<td>42</td>
<td>005F 0A</td>
<td>INC</td>
<td>R2</td>
</tr>
<tr>
<td>43</td>
<td>0060 D2A8</td>
<td>SETB</td>
<td>EX0 ; ENABLE DMA TRANSMIT INTERRUPT</td>
</tr>
<tr>
<td>44</td>
<td>0062 22</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0063</td>
<td>LOC_TMP</td>
<td>SET</td>
</tr>
<tr>
<td>51</td>
<td>0013</td>
<td>ORG</td>
<td>0013H</td>
</tr>
<tr>
<td>52</td>
<td>0013 020063</td>
<td>LJMP</td>
<td>RECEIVE_DMA</td>
</tr>
<tr>
<td>53</td>
<td>0063</td>
<td>ORG</td>
<td>LOC_TMP</td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td>RECEIVE_DMA:</td>
</tr>
</tbody>
</table>

19-5
0063 10000E 56 JBC  
0066 E0 59 MOVX  
0067 F6 60 MOV  
0068 08 61 INC  
0069 DA08 62 DJNZ  
006B D2CF 64 SETB  
006D D2CD 65 SETB  
006F D200 66 SETB  
0071 C2AA 67 CLR  
0073 32 69 L2: RETI  
0074 786A 71 L1: MOV  
0076 E0 73 MOVX  
0077 FA 74 MOV  
0078 32 75 RETI  
0079 77 LOC_TMP_SET $  
007A 78 ORG 0003H  
007B 020079 79 LJMP TRANSMIT_DMA  
0079 80 ORG LOC_TMP  
0082 786A 81 TRANSMIT_DMA  
0079 E7 84 MOV  
007A F0 85 MOVX  
007B 09 86 INC  
007C DA08 87 DJNZ  
007E C2A8 89 CLR  
0080 C294 90 CLR  
0082 D294 91 SETB  
0084 D2CE 92 SETB  
0086 32 94 L3: RETI  
0087 98 LOC_TMP_SET $  
0083 E7 84 MOV  
0084 F0 85 MOVX  
0085 09 86 INC  
0086 DA08 87 DJNZ  
008E C2A8 89 CLR  
0090 C294 90 CLR  
0092 D294 91 SETB  
0094 D2CE 92 SETB  
0096 32 94 L3: RETI  
0098 C2CC 94 XMIT: CLR  
0099 C2CC 95 XMIT: CLR  
009A 020056 107 LJMP ERROR  
009B 200067 100 LJMP SERIAL_INT  
009D 020087 101 ORG LOC_TMP  
009F 023 102 ORG 0023H  
009F 020087 100 LJMP SERIAL_INT  
00A1 020087 101 ORG LOC_TMP  
00A3 020056 107 LJMP ERROR  
00A6 200067 100 LJMP SERIAL_INT  
00A8 30CE06 105 JNB  
00A9 30CF0B 106 JNB  
00AA 200056 107 LJMP ERROR  
00AB 200067 100 LJMP SERIAL_INT  
00AD 30CE06 105 JNB  
00AE 30CF0B 106 JNB  
00AF 200056 107 LJMP ERROR  
00B0 200067 100 LJMP SERIAL_INT  
00B2 30CE06 105 JNB  
00B3 30CF0B 106 JNB  
00B4 200056 107 LJMP ERROR  
00B5 200067 100 LJMP SERIAL_INT  
00B7 30CE06 105 JNB  
00B8 30CF0B 106 JNB  
00B9 200056 107 LJMP ERROR  
00BA 200067 100 LJMP SERIAL_INT  
00BB 30CE06 105 JNB  
00BC 30CF0B 106 JNB  
00BD 200056 107 LJMP ERROR  
00BE 200067 100 LJMP SERIAL_INT  
00BF 30CE06 105 JNB  
00C0 30CF0B 106 JNB  
00C1 200056 107 LJMP ERROR  
00C2 200067 100 LJMP SERIAL_INT  
00C3 30CE06 105 JNB  
00C4 30CF0B 106 JNB  
00C5 200056 107 LJMP ERROR  
00C6 200067 100 LJMP SERIAL_INT  
00C7 30CE06 105 JNB  
00C8 30CF0B 106 JNB  
00C9 200056 107 LJMP ERROR  
00CA 200067 100 LJMP SERIAL_INT  
00CB 30CE06 105 JNB  
00CC 30CF0B 106 JNB  
00CD 200056 107 LJMP ERROR  
00CE 200067 100 LJMP SERIAL_INT  
00CF 30CE06 105 JNB  
00D0 30CF0B 106 JNB  
00D1 200056 107 LJMP ERROR  
00D2 200067 100 LJMP SERIAL_INT  
00D3 30CE06 105 JNB  
00D4 30CF0B 106 JNB  
00D5 200056 107 LJMP ERROR  
00D6 200067 100 LJMP SERIAL_INT  
00D7 30CE06 105 JNB  
00D8 30CF0B 106 JNB  
00D9 200056 107 LJMP ERROR  
00DA 200067 100 LJMP SERIAL_INT  
00DB 30CE06 105 JNB  
00DC 30CF0B 106 JNB  
00DD 200056 107 LJMP ERROR  
00DE 200067 100 LJMP SERIAL_INT  
00DF 30CE06 105 JNB  
00E0 30CF0B 106 JNB  
00E1 200056 107 LJMP ERROR  
00E2 200067 100 LJMP SERIAL_INT  
00E3 30CE06 105 JNB  
00E4 30CF0B 106 JNB  
00E5 200056 107 LJMP ERROR  
00E6 200067 100 LJMP SERIAL_INT  
00E7 30CE06 105 JNB  
00E8 30CF0B 106 JNB  
00E9 200056 107 LJMP ERROR  
00EA 200067 100 LJMP SERIAL_INT  
00EB 30CE06 105 JNB  
00EC 30CF0B 106 JNB  
00ED 200056 107 LJMP ERROR  
00EE 200067 100 LJMP SERIAL_INT  
00EF 30CE06 105 JNB  
00F0 30CF0B 106 JNB  
00F1 200056 107 LJMP ERROR  
00F2 200067 100 LJMP SERIAL_INT  
00F3 30CE06 105 JNB  
00F4 30CF0B 106 JNB  
00F5 200056 107 LJMP ERROR  
00F6 200067 100 LJMP SERIAL_INT  
00F7 30CE06 105 JNB  
00F8 30CF0B 106 JNB  
00F9 200056 107 LJMP ERROR  
00FA 200067 100 LJMP SERIAL_INT  
00FB 30CE06 105 JNB  
00FC 30CF0B 106 JNB  
00FD 200056 107 LJMP ERROR  
00FE 200067 100 LJMP SERIAL_INT  
00FF 30CE06 105 JNB 
009A D2AA 115 SETB EX1
009C 32 116 RETI
117
118 END

### SYMBOL TABLE LISTING

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>VALUE</th>
<th>ATTRIBUTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOV</td>
<td>B ADDR</td>
<td>00C8H.3</td>
<td>A</td>
</tr>
<tr>
<td>ERROR</td>
<td>C ADDR</td>
<td>0056H</td>
<td>A</td>
</tr>
<tr>
<td>EX0</td>
<td>B ADDR</td>
<td>00A8H.0</td>
<td>A</td>
</tr>
<tr>
<td>EX1</td>
<td>B ADDR</td>
<td>00A8H.2</td>
<td>A</td>
</tr>
<tr>
<td>FIRST_BYTE</td>
<td>B ADDR</td>
<td>0020H.0</td>
<td>A</td>
</tr>
<tr>
<td>IE</td>
<td>C ADDR</td>
<td>0026H</td>
<td>A</td>
</tr>
<tr>
<td>IP</td>
<td>D ADDR</td>
<td>00B8H</td>
<td>A</td>
</tr>
<tr>
<td>L1</td>
<td>C ADDR</td>
<td>0074H</td>
<td>A</td>
</tr>
<tr>
<td>L2</td>
<td>C ADDR</td>
<td>0073H</td>
<td>A</td>
</tr>
<tr>
<td>L3</td>
<td>C ADDR</td>
<td>0086H</td>
<td>A</td>
</tr>
<tr>
<td>LOC_TMP</td>
<td>C ADDR</td>
<td>0087H</td>
<td>A</td>
</tr>
<tr>
<td>PI</td>
<td>D ADDR</td>
<td>0090H</td>
<td>A</td>
</tr>
<tr>
<td>RBE</td>
<td>B ADDR</td>
<td>00C8H.6</td>
<td>A</td>
</tr>
<tr>
<td>RBL</td>
<td>D ADDR</td>
<td>00CBH</td>
<td>A</td>
</tr>
<tr>
<td>RBS</td>
<td>D ADDR</td>
<td>00CCCH</td>
<td>A</td>
</tr>
<tr>
<td>RCV</td>
<td>C ADDR</td>
<td>0090H</td>
<td>A</td>
</tr>
<tr>
<td>RECEIVE_DMA</td>
<td>C ADDR</td>
<td>0063H</td>
<td>A</td>
</tr>
<tr>
<td>RFL</td>
<td>D ADDR</td>
<td>00CDH</td>
<td>A</td>
</tr>
<tr>
<td>RTS</td>
<td>B ADDR</td>
<td>00C8H.5</td>
<td>A</td>
</tr>
<tr>
<td>SEND</td>
<td>C ADDR</td>
<td>0058H</td>
<td>A</td>
</tr>
<tr>
<td>SERIAL_INT</td>
<td>C ADDR</td>
<td>0087H</td>
<td>A</td>
</tr>
<tr>
<td>SI</td>
<td>B ADDR</td>
<td>00C8H.4</td>
<td>A</td>
</tr>
<tr>
<td>SMD</td>
<td>D ADDR</td>
<td>00C9H</td>
<td>A</td>
</tr>
<tr>
<td>SP</td>
<td>D ADDR</td>
<td>0081H</td>
<td>A</td>
</tr>
<tr>
<td>STAD</td>
<td>D ADDR</td>
<td>00CEH</td>
<td>A</td>
</tr>
<tr>
<td>TBF</td>
<td>B ADDR</td>
<td>00C8H.7</td>
<td>A</td>
</tr>
<tr>
<td>TBL</td>
<td>D ADDR</td>
<td>00DBH</td>
<td>A</td>
</tr>
<tr>
<td>TBS</td>
<td>D ADDR</td>
<td>00DCH</td>
<td>A</td>
</tr>
<tr>
<td>TCB</td>
<td>D ADDR</td>
<td>00DAH</td>
<td>A</td>
</tr>
<tr>
<td>TCON</td>
<td>D ADDR</td>
<td>0088H</td>
<td>A</td>
</tr>
<tr>
<td>TH1</td>
<td>D ADDR</td>
<td>008DH</td>
<td>A</td>
</tr>
<tr>
<td>TMOD</td>
<td>D ADDR</td>
<td>0089H</td>
<td>A</td>
</tr>
<tr>
<td>TRANSMIT_DMA</td>
<td>C ADDR</td>
<td>0079H</td>
<td>A</td>
</tr>
<tr>
<td>XMIT</td>
<td>C ADDR</td>
<td>0098H</td>
<td>A</td>
</tr>
</tbody>
</table>

REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8044

ASSEMBLY COMPLETE, NO ERRORS FOUND
Table 19-2. PL/M-86 Compiler Rupi/8088 Interface Example

SERIES-III PL/M-86 V1 0 COMILATION OF MODULE RUPI_BB
OBJECT MODULE PLACED IN: F1 RBB.OBJ
COMPILED INVOKED BY: PLM86 B6 'F1 RBB.SRC

```plaintext
*DEBUG
*TITLE ('RUPI/8088 INTERFACE EXAMPLE')
1   RUPI_BB:DO;
2   1  DECLARE
      LIT      LITERALLY 'LITERALLY',
      TRUE    LIT '01H',
      FALSE   LIT '00H',
      RECV_BUFFER(64) BYTE,
      XMIT_BUFFER(64) BYTE,
      I       BYTE,
      WAIT    BYTE.

   /* 8237 PORTS*/
   MASTER_CLEAR_37 LIT 'OFFDDH',
   COMMAND_37 LIT 'OFFDBH',
   ALL_MASK_37 LIT 'OFFDFH',
   SINGLE_MASK_37 LIT 'OFFDAH',
   STATUS_37 LIT 'OFFDBH',
   REQUEST_REG_37 LIT 'OFF9FH',
   MODE_REQ_37 LIT 'OFFDBH',
   CLEAR_BYTE_PTR_37 LIT 'OFFDCH',
   CHO_ADDR LIT 'OFFDDH',
   CHO_COUNT LIT 'OFFDH',
   CH1_ADDR LIT 'OFFDBH',
   CH1_COUNT LIT 'OFFD9H',
   CH2_ADDR LIT 'OFFD4H',
   CH2_COUNT LIT 'OFFDCH',
   CH3_ADDR LIT 'OFFDDH',
   CH3_COUNT LIT 'OFFD7H',

   /* 8237 BIT ASSIGNMENTS */
   CHO_SEL LIT '00H',
   CH1_SEL LIT '01H',
   CH2_SEL LIT '02H',
   CH3_SEL LIT '03H',
   WRITE_XFER LIT '04H',
   READ_XFER LIT '08H',
   DEMAND_MODE LIT '00H',
   SINGLE_MODE LIT '04H',
   BLOCK_MODE LIT '08H',
   SET_MASK LIT '04H',

   *EJECT
   /* 8259 PORTS */
   STATUS_POLL_59 LIT 'OFFEOH',
   ICW1_59 LIT 'OFFEOH',
   OCW1_59 LIT 'OFFEIH',
   OCW2_59 LIT 'OFFEOH',
   OCW3_59 LIT 'OFFEOH',
   ICW2_59 LIT 'OFFEIH',
   ICW3_59 LIT 'OFFEIH',
   ICW4_59 LIT 'OFFEIH',

   /* INTERRUPT SERVICE ROUTINE */
   3 1   OFF_RECV_DMA: PROCEDURE INTERRUPT 32;
   4 2   OUTPUT(SINGLE_MASK_37)='40H';
   5 2   WAIT=FALSE;
   6 2   END;
```

19-8
7 1 DISABLE;

    /* INITIALIZE 8237 */
8 1 OUTPUT(MASTER_CLEAR_37) =0;
9 1 OUTPUT(COMMAND_37) =040H;
10 1 OUTPUT(ALL_MASK_37) =0FH;
11 1 OUTPUT(MODE_REQ_37) = (SINGLE_MODE OR WRITE_XFER OR CHO_SEL);
12 1 OUTPUT(MODE_REQ_37) = (SINGLE_MODE OR READ_XFER OR CH1_SEL);
13 1 OUTPUT(CLEAR_BYTE_PTR_37) =0;
14 1 OUTPUT(CH0_ADDR) =00H;
15 1 OUTPUT(CH0_ADDR) =40H;
16 1 OUTPUT(CH0_COUNT) =64;
17 1 OUTPUT(CH0_COUNT) =00;
18 1 OUTPUT(CH1_ADDR) =40H;
19 1 OUTPUT(CH1_ADDR) =40H;
20 1 OUTPUT(CH1_COUNT) =64;
21 1 OUTPUT(CH1_COUNT) =00;

    /* INITIALIZE 8259 */
22 1 OUTPUT(ICW1_59) =13H; /*SINGLE MODE, EDGE TRIGGERED*/
23 1 OUTPUT(ICW2_59) =20H; /*INTERRUPT TYPE 32*/
24 1 OUTPUT(ICW4_59) =03H; /*AUTO-EOI*/
25 1 OUTPUT(ICW1_59) =0FH; /*ENABLE INTERRUPT LEVEL 0*/

*EXIT
26 1 CALL SET_INTERRUPT (32, OFF_RECV_DMA); /*LOAD INTERRUPT VECTOR LOCATION*/

27 1 XMIT_BUFFER(0)=64; /*THE FIRST BYTE IN THE BLOCK OF DATA IS THE NUMBER OF BYTES TO BE TRANSFERRED NOT INCLUDING THE FIRST BYTE*/

28 1 DO I=1 TO 64; /*FILL UP THE XMIT_BUFFER WITH DATA*/
29 2 XMIT_BUFFER(I)=1;
30 2 END;
31 1 OUTPUT(ALL_MASK_37)=0FH; /*ENABLE CHANNEL 1 AND 2*/
32 1 ENABLE;
33 1 WAIT=TRUE;
34 1 DO WHILE WAIT;
35 2 END; /*A BLOCK OF DATA WILL BE TRANSFERRED TO THE RUP1.
WHEN THE RUP1 RECEIVES A BLOCK OF DATA IT WILL SEND IT TO THE BOSS MEMORY AND INTERRUPT THE BOSS.
The INTERRUPT SERVICE ROUTINE WILL SHUT OFF THE DMA CONTROLLER AND SET 'WAIT' FALSE*/
36 1 DO WHILE 1;
37 2 END;
38 1 END;

MODULE INFORMATION:

| CODE AREA SIZE | 00D7H | 215D |
| CONSTANT AREA SIZE | 0000H | 0D |
| VARIABLE AREA SIZE | 0082H | 130D |
| MAXIMUM STACK SIZE | 001EH | 30D |
124 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS

END OF PL/I-B6 COMPILATION
A HIGH PERFORMANCE NETWORK
USING THE 8044

19.2.1 Introduction
This section describes the design of an SDLC data link using the 8044 (RUII) to implement a primary station and a secondary station. The design was implemented and tested. The following discussion assumes that the reader understands the 8044 and SDLC. This section is divided into two parts. First the data link design example is discussed. Second the software modules used to implement the data link are described. To help the reader understand the discussion of the software, flow charts and software listings are displayed in Appendix A and Appendix B, respectively.

Application Description
This particular data link design example uses a two wire half-duplex multidrop topology as shown in figure 19-4. In an SDLC multidrop topology the primary station communicates with each secondary station. The secondary stations communicate only to the primary. Because of this hierarchial architecture, the logical topology for an SDLC multidrop is a star as shown in figure 19-5. Although the physical topology of this data link is multidrop, the easiest way to understand the information flow is to think of the logical (star) topology. The term data link in this case refers to the logical communication pathways between the primary station and the secondary stations. The data links are shown in figure 19-5 as two way arrows.

The application example uses dumb async terminals to interface to the SDLC network. Each secondary station has an async terminal connected to it. The secondary stations are in effect protocol converters which allows any async terminal to communicate with any other async terminal on the network. The secondary stations use an 8044 with a UART to convert SDLC to async. Figure 19-6 displays a block diagram of the data link. The primary station, controls the data link. In addition to data link control the primary provides a higher level layer which is a path control function or networking layer. The primary serves as a message exchange or switch. It receives information from one secondary station and retransmits it to another secondary station. Thus a virtual end to end connection is made between any two secondary stations on the network.

Three separate software modules were written for this network. The first module is a Secondary Station Driver (SSD) which provides an SDLC data link interface and a user interface. This module is a general purpose driver which requires application software to run it. The user interface to the driver provides four functions: OPEN, CLOSE, TRANSMIT, and SIU__RECV. Using these four functions properly will allow any application software to communicate over this SDLC data link without knowing the details of SDLC. The secondary station driver uses the 8044's AUTO mode.

The second module is an example of application software which is linked to the secondary station driver. This module drives the 8251A, buffers data, and interfaces with the secondary station driver's user interface.

The third module is a primary station, which is a stand-alone program (i.e., it is not linked to any other module). The primary station uses the 8044's NON-AUTO or FLEXIBLE mode. In addition to controlling the data link it acts as a message switch. Each time a secondary station transmits a frame, it places the destination address of the frame in the first byte of the information or I field. When the primary station receives a frame, it removes the first byte in the I field and retransmits the frame to the secondary station whose address matches this byte.

This network provides two complete layers of the OSI (Open Systems Interconnection) reference model: the physical layer and the data link layer. The physical layer implementation uses the RS-422 electrical interface. The mechanical medium consists of ribbon cable and connectors. The data link layer is defined by SDLC. SDLC's use of acknowledgements and frame numbering guarantees that messages will be received in the same order in which they were sent. It also guarantees message integrity over the data link. However this network will not guarantee secondary to secondary message delivery, since there are acknowledgements between secondary stations.

19.2.2 Hardware
The schematic of the hardware is given in figure 19-7. The 8251A is used as an async communications controller, in support of the 8044. TxRDY and RxRDY on the 8251A are both tied to the two available external interrupts of the 8044 since the secondary station driver is totally interrupt driven. The 8044 buffers the data and some variables in a 2016 (2K x 8 static RAM). The 8254 programmable interval timer is employed as a programmable baud rate generator and system clock driver for the 8251A. The third output from the 8254 could be used as an external baud rate generator for the 8044. The 2732A shown in the diagram was not used since the software for both the primary and secondary stations used far less than the 4 Kbytes provided on the 8744. For the async interface, the standard RS-232
mechanical and electrical interface was used. For the SDLC channel, a standard two wire three state RS-422 driver is used. A DIP switch connected to one of the available ports on the 8044 allows the baud rate, parity, and stop bits to be changed on the async interface. The primary station hardware does not use the USART, 8254, nor the RS-232 drivers.

19.2.3 SDLC Basic Repertoire
The SDLC commands and responses implemented in the data link include the SDLC Basic Repertoire as defined in the IBM SDLC General Information manual. Table 19-3 shows the commands and responses that the primary and the secondary station in this data link design recognize and send.
Figure 19-6. Block Diagram of the Data Link Application Example
Figure 19-7. Schematic of Async/SDLC Secondary Station Protocol Converter
Table 19-3. Data Link Commands and Responses Implemented for This Design

<table>
<thead>
<tr>
<th>PRIMARY STATION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Responses</strong></td>
</tr>
<tr>
<td><strong>Recognized</strong></td>
</tr>
<tr>
<td>Unnumbered</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Supervisory</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECONDARY STATION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commands</strong></td>
</tr>
<tr>
<td><strong>Recognized</strong></td>
</tr>
<tr>
<td>Unnumbered</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Supervisory</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Information</td>
</tr>
</tbody>
</table>

*not included in the SDLC Basic Repertoire

The term command specifically means all frames which the primary station transmits and the secondary stations receive. Response refers to frames which the secondary stations transmit and the primary station receives.

**Number of Outstanding Frames**

This particular data link design only allows one outstanding frame before it must receive an acknowledgement. Immediate acknowledgement allows the secondary station drivers to use the AUTO mode. In addition, one outstanding frame uses less memory for buffering, and the software becomes easier to manage.

**19.2.4 Secondary Station Driver using AUTO mode**

The 8044 secondary station driver (SSD) was written as a general purpose SDLC driver. It was written to be linked to an application module. The application software implements the actual application in addition to interfacing to the SSD. The main application could be, a printer or plotter, a medical instrument, or a terminal. The SSD is independent of the main application, it just provides the SDLC communications. Existing 8051 applications could add high performance SDLC communications capability by linking the SSD to the existing software and providing additional software to be able to communicate with the SSD.

**Data Link Interface and User Interface States**

The SSD has two software interfaces: a data link interface and a user interface as show in Figure 19-8. The data link interface is the part of the software which controls the SDLC communications. It handles link access, command recognition/response, acknowledgements, and error recovery. The user interface provides four functions: OPEN, CLOSE, TRANSMIT, and SIU_RECV. These are the only four functions which the application software has to interface in order to communicate using SDLC. These four functions are common to many I/O drivers like floppy and hard disks, keyboard/CRT, and async communication drivers.

The data link and the user interface each have their own states. Each interface can only be in one state at any time. The SSD uses the states of these two interfaces to help synchronize the application module to the data link.

There are three states which the secondary station data link interface can be in: Logical Disconnect State (L_D_S), Frame Reject State (FRMR_S), and Information Transfer State (I_T_S). The Logical Disconnect State is when a station is physically connected to the channel but either the primary or secondary have not agreed to enter the Information Transfer State. Both the primary and the secondary stations synchronize to enter into the Information Transfer State. Only when the secondary station is in the I_T_S is it able to transfer data or information to the primary. The Frame Reject State (FRMR_S) indicates that the secondary station has lost software synchronization with the primary or encountered some kind of error condition. When the secondary station is in the FRMR_S, the primary station must reset the secondary to resynchronize.

The user interface has two states, open or closed. In the closed state the user program does not want to communicate over the network. The communications channel is closed and not available for use. The secondary station tells the primary this by responding to all commands with DM. The primary continues to poll the secondary in case it wants to enter the I_T_S state. When the user program begins communication over the data link it goes into the open state. It does this by calling the OPEN procedure. When the user interface is in the open state it may transfer information to the primary.
SECONDARY STATION

APPLICATION MODULE

SECONDARY STATION DRIVER MODULE

DATA LINK INTERFACE

USER INTERFACE

USER STATES

1. OPEN
2. CLOSED

DATA LINK STATES

1. LOGICAL DISCONNECT STATE
2. INFORMATION TRANSFER STATE
3. FRAME REJECT STATE

SSD INTERFACE

OPEN CLOSE TRANSMIT SIU RECV

SSD INTERFACE PROCEDURES

Figure 19-8. Secondary Station Software Modules
Secondary Stations Commands, Responses and State Transitions

Table 19-4 shows the commands which the secondary station recognizes and the responses it generates. The first row in table 19-4 displays commands the secondary station recognizes and each column shows the potential responses with respect to secondary station. For example, if the secondary is in the Logical Disconnect State it will only respond with DM, unless it receives a SNRM command and the user state is open. If this is the case, then the response will be UA and the secondary station will move into the I_T_S.

Figure 19-9 shows the state diagram of the secondary station. When power is first applied to the secondary station, it goes into the Logical Disconnect State. As mentioned above, the I_T_S is entered when the secondary station receives a SNRM command and the user state is open. The secondary responds with UA to let the primary know that it has accepted the SNRM and is entering the I_T_S. The I_T_S can go into either the L_D_S or the FRMR_S. The I_T_S goes into the L_D_S if the primary sends the secondary DISC. The secondary has to respond with UA, and then goes into the L_D_S. If the user interface changes from open to close state, then the secondary sends RD. This causes the primary to send a DISC.

The FRMR_S is entered when a secondary station is in the L_D_S and either one of the following conditions occurs:

- A command can not be recognized by the secondary station.
- There is a buffer overrun.
- The Nr that was received from the primary station is invalid.

The secondary station cannot leave the FRMR_S until it receives a SNRM or a DISC command.

Software description of the SSD

To aid in following the description of the software, the reader may either look at the flow charts which are given for each procedure, or read the PL/M-51 listing provided in Appendix A.

A block diagram of the software structure of the SSD is given in figure 19-10. A complete module is identified by the dotted box, and a procedure is identified by the solid box. Therefore the SIU_RECV procedure is not included in the SSD module, it exists in the application software. Two or more procedures connected by a solid line means the procedure above calls the procedure below. Transmit, Power_on_D, Close, and Open are all called by the application software. Procedures without any solid lines connected above are interrupt procedures. The only interrupt procedure in the SSD module is the SIU_INT.

The entire SSD module is interrupt driven. Its design allows the application program could handle real time events or just dedicate more CPU time to the application program. The SIU_INT is the only interrupt pro-

---

Table 19-4 . Secondary Station Responses to Primary Station Commands

<table>
<thead>
<tr>
<th>Data Link States</th>
<th>I</th>
<th>RR</th>
<th>RNR</th>
<th>SNRM</th>
<th>DISC</th>
<th>TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information transfer state</td>
<td></td>
<td>I</td>
<td></td>
<td></td>
<td>I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RR</td>
<td>RR</td>
<td>RR</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RNR</td>
<td>RNR</td>
<td>RNR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RD</td>
<td>RD</td>
<td>RD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FRMR</td>
<td>FRMR</td>
<td>FRMR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical disconnect state</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
</tr>
<tr>
<td>Frame reject state</td>
<td>FRMR</td>
<td>FRMR</td>
<td>FRMR</td>
<td></td>
<td></td>
<td>FRMR</td>
</tr>
</tbody>
</table>

---

19-16
Figure 19-9. State Diagram of Secondary Station
Figure 19-10. Secondary Station Driver
procedure in the SSD. It is automatically entered when an SIU interrupt occurs. This particular interrupt can be the lowest priority interrupt in the system.

**SSD Initialization**

Upon reset the application software is entered first. The application software initializes its own variables then calls Power_ On_D which is the SSD's initialization routine. The SSD's initialization sets up the transmit and receive data buffer pointers (TBS and RBS), the receive buffer length (RBL), and loads the State variables. The STATION_STATE begins in the L__D__S state, and the USER__STATE begins in the closed state. Finally Power_ On_D initializes XMIT__BUFFER__EMPTY which is a bit flag. This flag serves as a semaphore between the SSD and the application software to indicate the status of the on chip transmit buffer. The SSD does not set the station address. It is the application software’s responsibility to do this. After initialization, the SSD is ready to respond to all of the primary stations commands. Each time a frame is received with a matching station address and a good CRC, the SIU__INT procedure is entered.

**SIU__INT Procedure**

The first thing the SIU__INT procedure clears the serial interrupt __bit (SI) in the STS register. If the SIU__INT procedure returns with this bit set, another SI interrupt will occur.

The SIU__INT procedure is branches three independent cases. The first case is entered if the STATION__STATE is not in the L__T__S. If this is true, then the SIU is not in the AUTO mode, and the CPU will have to respond to the primary on its own. (Remember that the AUTO mode is entered when the STATION__STATE enters into L__T__S.) If the STATION__STATE is in the L__T__S, then either the SIU has just left the AUTO mode, or is still in the AUTO mode. This is the second and third case respectively.

In the first case, if the STATION__STATE is not in the L__T__S, then it must be in either the L__D__S or the FRMR__S. In either case a separate procedure is called based on which state the station is in. The In__Disconnect_ State procedure sends to the primary a DM response, unless it received a SNRM command and the USER__STATE equals open. In that case the SIU sends an UA and enters into the L__T__S. The In__FRMR State procedure will send the primary the FRMR response unless it received either a DISC or an SNRM. If the primary’s command was a DISC, then the secondary will send an UA and enter into the L__D__S. If the primary’s command was a SNRM, then the secondary will send an UA, enter into the L__T__S, and clear NSNR register.

For the second case, if the STATION__STATE is in the L__T__S but the SIU left the AUTO mode, then the CPU must determine why the AUTO mode was exited, and generate a response to the primary. There are four reasons for the SIU to automatically leave the AUTO mode. The following is a list of these reasons, and the responses given by the SSD based on each reason.

1. The SIU has received a command field it does not recognize.
   Response: If the CPU recognizes the command, it generates the appropriate response. If neither the SIU nor the CPU recognize the command, then a FRMR response is sent.

2. The SIU has received a Sequence Error Sent (SES=1 in NSNR register). Nr(P) ≠ Nr(S)+1, and Nr(P) ≠ Nr(S).
   Response: Send FRMR.

3. A buffer overrun has occurred. BOV=1 in STS register.
   Response: Send FRMR.

4. An I frame with data was received while RPB=1.
   Response: Go back into AUTO mode and send an AUTO mode response.

In addition to the above reasons, there is one condition where the CPU forces the SIU out of the AUTO mode. This is discussed in the SSD’s User Interface Procedures section in the CLOSED procedure description.

Finally, case three is when the STATION__STATE is in the L__T__S and the AUTO mode. The CPU first looks at the TBF bit. If this bit is 0 then the interrupt may have been caused by a frame which was transmitted and acknowledged. Therefore the XMIT__BUFFER__EMPTY flag is set again indicating that the application software can transmit another frame.

The other reason this section of code could be entered is if a valid I frame was received. When a good I frame is received the RBE bit equals 0. This means that the receiver is disabled. If the primary were to poll the 8044 while RBE=0, it would time out since no response would be given. Time outs reduce network throughput. To improve network performance, the CPU first sets RBP, then sets RBE. Now when the primary polls the 8044 an immediate RNR response is given. At this point the SSD calls the application software procedure SIU__RECV and passes the length of the data as a parameter. The SIU__RECV procedure reads the data out of the receive buffer then returns to the SSD module. Now that the receive information has been transferred, RBP can be cleared.

**Command_Decode Procedure**

The Command_Decode procedure is called from the SIU__INT procedure when the STATION__STATE = I__T__S and the SIU left the AUTO mode as a result.
of not being able to recognize the receive control byte. Commands which the SIU AUTO mode does not recognize are handled here. The commands recognized in this procedure are: SNRM, DISC, and TEST. Any other command received will generate a Frame Reject with the nonimplemented command bit set in the third data byte of the FRMR frame. Any additional unnumbered frame commands which the secondary station is going to implement, should be implemented in this procedure.

If a SNRM is received the command_decode procedure calls the SNRM_Response procedure. The SNRM_Response procedure sets the STATION_STATE = I_T_S, clears the NSNR register and responds with an UA frame. If a DISC is received, the command_decode procedure sets the STATION_STATE = L_D_S, and responds with an UA frame. When a TEST frame is received, and there is no buffer overrun, the command_decode procedure responds with a TEST frame retransmitting the same data it received. However if a TEST frame is received and there is a buffer overrun, then a TEST frame will be sent without any data, instead of a FRMR with the buffer overrun bit set.

Frame Reject Procedures

There are two procedures which handle the FRMR state: XMIT_FRMR and IN_FRMR_STATE. XMIT_FRMR is entered when the secondary station first goes into the FRMR state. The frame reject response frame contains the FRMR response in the command field plus three additional data bytes in the I field. Figure 19-11 displays the format for the three data byte in the I field of a FRMR response. The XMIT_FRMR procedure sets up the Frame Reject response frame based on the parameter REASON which is passed to it. Each place in the SSD code that calls the XMIT_FRMR procedure, passes the REASON that this procedure was called, which in turn is communicated to the primary station. The XMIT_FRMR procedure uses three bytes of internal RAM which it initializes for the correct response. The TBS and TBL registers are then changed to point to the FRMR buffer so that when a response is sent these three bytes will be included in the I field.

The IN_FRMR_STATE procedure is called by the SIU_INT procedure when the STATION_STATE already is in the FRMR state and a response is required. The IN_FRMR_STATE procedure will only allow two commands to remove the secondary station from the FRMR state: SNRM and DISC. Any other command which is received while in the FRMR state will result a FRMR response frame.

XMIT_UNNUMBERED Procedure

This is a general purpose transmit procedure, used only in the FLEXIBLE mode, which sends unnumbered responses to the primary. It accepts the control byte as a parameter, and also expects the TBL register to be set before the procedure is called. This procedure waits until the frame has been transmitted before returning. If this procedure returned before the transmit interrupt was generated, the SIU_INT routine would be entered. The SIU_INT routine would not be able to distinguish this condition.
SSD's User Interface Procedures -- OPEN, CLOSE, TRANSMIT, SIU_RECV -- are discussed in the following section.

The OPEN procedure is the simplest of all, it changes the USER_STATE to OPEN_S then returns. This lets the SSD know that the user wants to open the channel for communications. When the SSD receives a SNRM command, it checks the USER_STATE. If the USER_STATE is open, then the SSD will respond with an UA, and the STATION_STATE enters the I_T_S.

The CLOSE procedure is also simple, it changes the USER_STATE to CLOSED_S and sets the AM bit to 0. Note that when the CPU sets the AM bit to 0 it puts the SIU out of the AUTO mode. This event is asynchronous to the events on the network. As a result an I frame can be lost. This is what can happen.

1. AM is set to 0 by the CLOSE Procedure.
2. An I frame is received and a S1 interrupt occurs.
3. The SIU_INT procedure enters case 2. (STATION_STATE = I_T_S, and AM = 0)
4. Case 2 detects that the USER_STATE = CLOSED_S, sends a RD response and ignores the fact that an I frame was received.

Therefore it is advised to never call the CLOSE procedure or take the SIU out of the AUTO mode when it is receiving I frames or an I frame will be lost.

For both the TRANSMIT and SIU_RECV procedures, it is the application software's job to put data into the transmit buffer, and take data out of the receive buffer. The SSD does not transfer data in or out of its transmit or receive buffers because it does not know what kind of buffering the application software is implementing. What the SSD does do is notify the application software when the transmit buffer is empty, XMIT_BUFFER_EMPTY = 1, and when the receive buffer is full.

One of the functions that the SSD performs to synchronize the application software to the SDLC data link. However some of the synchronization must also be done by the application software. Remember that the SSD does not want to hang up the application software waiting for some event to occur on the SDLC data link, therefore the SSD always returns to the application software as soon as possible.

For example, when the application software calls the OPEN procedure, the SSD returns immediately. The application software thinks that the SDLC channel is now open and it can transmit. This is not the case. For the channel to be open, the SSD must receive a SNRM from the primary and respond with a UA. However, the SSD does not want to hang up the application software waiting for a SNRM from the primary before returning from the OPEN procedure. When the TRANSMIT procedure is called, the SSD expects the STATION_STATE to be in the I_T_S. If it isn't, the SSD refuses to transmit the data. The TRANSMIT procedure first checks to see if the USER_STATE is open, if not the USER_STATE_CLOSED parameter is passed back to the application module. The next thing TRANSMIT checks is the STATION_STATE. If this is not open, then TRANSMIT passes back LINK_DISCONNECTED. This means that the USER_STATE is open, but the SSD hasn't received a SNRM command from the primary yet. Therefore, the application software should wait awhile and try again. Based on network performance, one knows the maximum amount of time it will take for a station to be polled. If the application software waits this length of time and tries again but still gets a LINK_DISCONNECTED parameter passed back, higher level recovery must be implemented.

Before loading the transmit buffer and calling the TRANSMIT procedure, the application software must check to see that XMIT_BUFFER.EMPTY = 1. This flag tells the application software that it can write new data into the transmit buffer and call the TRANSMIT procedure. After the application software has verified that XMIT_BUFFER.EMPTY = 1, it fills the transmit buffer with the data and calls the TRANSMIT procedure passing the length of the buffer as a parameter. The TRANSMIT procedure checks for three reasons why it might not be able to transmit the frame. If any of these three reasons are true, the TRANSMIT procedure returns a parameter explaining why it couldn't send the frame. If the application software receives one of these responses, it must rectify the problem and try again. Assuming these three conditions are false, then the SSD clears XMIT_BUFFER.EMPTY, attempts to send the data and returns the parameter DATA_TRANSMITTED. XMIT_BUFFER.EMPTY will not be set to 1 again until the data has been transmitted and acknowledged.

The SIU_RECV procedure must be incorporated into the application software module. When a valid I frame is received by the SIU, it calls the SIU_RECV procedure and passes the length of the received data as a parameter. The SIU_RECV procedure must remove all of the data from the receive buffer before returning to the SIU_INT procedure.

**Linking up to the SSD**

Figure 19-12 shows necessary parts to include in a PL/M-51 application program that will be linked to the SSD module. RL51 is used to link and locate the SSD and application modules. The command line used to do this is:
Set the control, can be used to ensure that.

Another purpose is to implement and test a practical application. This application software performs I/O with an async terminal through a USART, buffers data, and also performs I/O with the SSD. In addition, it allows the user on the async terminal to: set the station address, set the destination address, and go online and offline. Setting the station address sets the byte in the STD register. The destination address is the first byte in the I field. Going online or offline results in either calling the OPEN or CLOSE procedure respectively.

After the secondary station powers up, it enters the ‘terminal mode’, which accepts data from the terminal. However, before any data is sent, the user must configure the station. The station address and destination address must be set, and the station must be placed online. To configure the station the ESC character is entered at the terminal which puts the protocol converter into the ‘configure mode’. Figure 20-13 shows the menu which appears on the terminal screen.

1 - Set the Station Address
2 - Set the Destination Address
3 - Go Online
4 - Go Offline
5 - Return to terminal mode
   Enter option __

**Figure 19-13. Menu for the Protocol Converter**

In the terminal mode data is buffered up in the secondary station. A Line Feed character ‘LF’ tells the secondary station to send an I frame. If more than 60 bytes are buffered in the secondary station when a ‘LF’ is received, the applications software packetizes the data into 60 bytes or less per frame. If a LF is entered when the station is offline, an error message comes on the screen which says ‘Unable to Get Online’.

The secondary station also does error checking on the async interface for Parity, Framing Error, and Overrun Error. If one of these errors are detected, an error message is displayed on the terminal screen.

**Buffering**

There are two separate buffers in the application module: a transmit buffer and a receive buffer. The transmit buffer receives data from the USART, and sends data to the SSD. The receive buffer receives data from the SSD, and transmits data to the USART. Each buffer is a 256 byte software FIFO. If the transmit FIFO becomes full and no ‘LF’ character is received, the secondary station automatically begins sending the data. In addition, the application module will shut off the terminal’s transmitter using CTS until the FIFO has been partially emptied. A block diagram of the buffering for the protocol converter is given in Figure 19-14.

**Application Module Software**

A block diagram of the application module software is given in Figure 19-15. There are three interrupt
Figure 19-14. Block Diagram of Secondary Station Protocol Converter Illustrating Buffering
functions in this module: USART_RECV_INT, USART_XMIT_INT, and TIMER_0_INT. The first two are for servicing the USART. TIMER_0_INT is used if the TRANSMIT procedure in the SSD is called and does not return with the DATA_TRANSMITTED parameter. TIMER_0_INT employs Timer 0 to wait a finite amount of time before trying to transmit again. The highest priority interrupt is USART_RECV_INT. The main program and all the procedures it calls use register bank 0, USART_XMIT_INT and TIMER_0_INT and FIFO_R_OUT use bank 1, while USART_RECV_INT and all the procedures it calls use register bank 2.

**Power_On Procedure**

The Power_On procedure initializes all of the chips in the system including the 8044. The 8044 is initialized to use the on-chip DPLL with NRZI coding, Preframe Sync, and Timer 1 auto-reload at a baud rate of 62.5 Kbps. The 8254 and the 8251A are initialized next based on the DIP switch values attached to port 1 on the 8044. Variables and pointers are initialized, then the SSD’s Power-Up Procedure, Power_On_D, is called. Finally, the interrupt system is enabled and the main program is entered.

**Main Program**

The main program is a simple loop which waits for a frame transmit command. A frame transmit command is indicated when the variable SEND_DATA is greater than 0. The value of SEND_DATA equals the number of ‘LF’ characters in the transmit FIFO, hence it also indicates the number of frames pending transmission. Each time a frame is sent, SEND_DATA is decremented by one. When the SEND_DATA is greater than 0, the main program falls down into the next loop which polls the XMIT_BUFFER EMPTY bit. When SEND_DATA = 0, the SIU_XMIT_BUFFER can be loaded. The first byte in the buffer is loaded with the destination address while the rest of the buffer is loaded with the data. Bytes are removed from the transmit FIFO and placed into the SIU_XMIT_BUFFER until one of three things happen: 1. a ‘LF’ character is read out of the FIFO, 2. the number of bytes loaded equals the size of the SIU_XMIT_BUFFER, or 3. the transmit FIFO is empty.

After the SIU_XMIT_BUFFER is filled, the SSD TRANSMIT procedure is called and the results from the procedure are checked. Any result other than DATA_TRANSMITTED will result in several retries within a finite amount of time. If all the retries fail then the LINK_DISC procedure is called which sends a message to the terminal, ‘Unable to Get Online’.

**USART_RECV_INT Procedure**

When the 8251A receives a character, the RxRDY pin on the 8251A is activated, and this interrupt procedure is entered. The routine reads the USART status register to determine if there are any errors in the character received. If there are, the character is discarded and the ERROR procedure is called which prints the type of error on the screen. If there are no errors, the received character is checked to see if it’s an ESC. If it is an ESC, the MENU procedure is called which allows the user to change the configuration. If neither one of these two conditions exits the received character is inserted into the transmit FIFO. The received character may or may not be echoed back to the terminal based on the dip switch settings.

**Transmit FIFO**

The transmit FIFO consists of two procedures: FIFO_T_IN and FIFO_T_OUT. FIFO_T_IN inserts a character into the FIFO and FIFO_T_OUT removes a character from the FIFO. The FIFO itself is an array of 256 bytes called FIFO_T. There are two pointers used as indexes in the array to address the characters: IN_PTR_T and OUT_PTR_T. IN_PTR_T points to the location in the array which will store the next byte of data inserted. OUT_PTR_T points to the next byte of data removed from the array. Both IN_PTR_T and OUT_PTR_T are declared as bytes. The FIFO_T_IN procedure receives a character from the USART_RECV_INT procedure and stores it in the array location pointed to by IN_PTR_T. Then, IN_PTR_T is incremented. Similarly, when FIFO_T_OUT is called by the main program, to load the SIU_XMIT_BUFFER, the byte in the array pointed to by OUT_PTR_T is read, then OUT_PTR_T is incremented. Since IN_PTR_T and OUT_PTR_T are always incremented, they must be able to roll over when they hit the top of the 256 byte address space. This is done automatically by having both IN_PTR_T and OUT_PTR_T declared as bytes. Each character inserted into the transmit FIFO is tested to see if it’s a LF. If it is a LF, the variable SEND_DATA is incremented which lets the main program know that it is time to send an I frame. Similarly each character removed from the FIFO is tested. SEND_DATA is decremented for every LF character removed from the FIFO.

IN_PTR_T and OUT_PTR_T are also used to indicate how many bytes are in the FIFO, and whether it is full or empty. When a character is placed into the FIFO and IN_PTR_T is incremented, the FIFO is full if IN_PTR_T equals OUT_PTR_T. When a character is read from the FIFO and OUT_PTR_T is incremented, the FIFO is empty if OUT_PTR_T equals IN_PTR_T. If the FIFO is neither full nor empty, then it is in use. A byte called BUFFER_STATUS is used to indicate one of these three conditions. The application module uses the buffer status information to control the flow of data into and out of the FIFO. When the transmit FIFO is empty, the main program must stop loading bytes into the SIU_...
Figure 19-15: Block Diagram of User Software
controls link access, link level error recovery, and the

enabled while the receive

can be used as the basis for this primary station design.

design would extensively off-load link control functions
or front end processors

data. In the meantime, the

ing to all polls from the primary with a RNR supervisory

ing the primary station

front end processor for a microprocessor, implemen­
ting the

flow of information. Secondaries can only transmit

data link.

The primary station must know the addresses of all the
stations which will be on the network. The software for
this primary needs to know this before it is compiled,
however a more flexible system would down load these
parameters.

From the listing of the software it can be seen that the
variable NUMBER_OF_STATATIONS is a literal
declaration, which is 2 in this design example. There
were three stations tested on this data link, two secon­
daries and one primary. Following the NUMBER_OF_STATATIONS declaration is a table, loaded into the
object code file at compile time, which lists the addresses
of each secondary station on the network.

Remote Station Database

The primary station keeps a record of each secondary
station on the network. This is called the Remote
Station Database (RSD). The RSD in this software is
an array of structures, which can be found in the listing
and also in Figure 19-16. Each RSD stores the necessary
information about that secondary station.

To add additional secondary stations to the network,
one simply adjusts the NUMBER_OF_STATATIONS
declaration, and adds the additional addresses to the
SECONDARY_ADDRESSES table. The number of
RSDs is automatically allocated at compile time, and
the primary automatically polls each station whose
address is in the SECONDARY_ADDRESSES table.

Memory for the RSDs resides in external RAM. Based
on memory requirements for each RSD, the maximum
number of stations can be easily buffered in external
RAM. (254 secondary stations is the maximum number
SDLC will address on the data link; i.e. 8 bit address,
FFH is the broadcast address, and 0 is the null address.
Each RSD uses 70 bytes of RAM. 70 x 254 = 17,780.)

The station state, in the RSD structure, maintain the
status of the secondary. If this byte indicates that the
secondary is in the DISCONNECT_S, then the primary
tries to put the station in the I_T_S by sending a
SNRM. If the response is a UA then the station state
changes into the I_T_S. Any other frame received
results in the station state remaining in the
DISCONNECT_S. When the RSD indicates that the
station state is in the I_T_S, the primary will send
either a I, RR, or RNR command, depending on the
local and remote buffer status. When the station state
equals GO_TO_DISC the primary will send a DISC
command. If the response is an UA frame, the station
state will change to DISCONNECT_S, else the station
state will remain in GO_TO_DISC. The station state
is set to GO_TO_DISC when one of the following
responses occur:

1. A receive buffer overrun in the primary.
2. An I frame is received and Nr(P) ≠Ns(S).
3. An I frame or a Supervisory frame is received and
Ns(P) + 1 ≠ Nr(S) andNs(P) ≠Nr(S).
4. A FRMR response is received.
5. An RD response is received.
6. An unknown response is received.

The send count (Ns) and receive count (Nr) are also
maintained in the RSD. Each time an I frame is sent
by the primary and acknowledged by the secondary, Ns
is incremented. Nr is incremented each time a valid I
frame is received. BUFFER_STATUS indicates the
status of the secondary stations buffer. If a RR response
is received, BUFFER_STATUS is set to BUFFER_
READY. If a RNR response is received, BUFFER__STATUS is set to BUFFER_NOT_READY.

Buffering

The buffering for the primary station is as follows: within each, RSD is a 64 byte array buffer which is initially empty. When the primary receives an I frame, it looks for a match between the first byte of the I frame and the addresses of the secondaries on the network. If a match exists, the primary places the data in the RSD buffer of the destination station. The INFO.LENGTH in the RSD indicates how many bytes are in the buffer. If INFO.LENGTH equals 0, then the buffer is empty. The primary can buffer only one I frame per station. If a second I frame is received while the addressed secondary’s RSD buffer is full, the primary cannot receive any more I frames. At this point the primary continues to poll the secondaries using RNR supervisory frame.

Primary Station Software

A block diagram of the primary station software is shown in Figure 19-17. The primary station software consists of a main program, one interrupt routine, and several procedures. The POWER.ON procedure begins by initializing the SIU’s DMA and enabling the receiver. Then each RSD is initialized. The DPLL and the timers are set, and finally the TIMER 0 interrupt is enabled.

The main program consists of an iterative do loop within a do forever loop. The iterative do loop polls each secondary station once through the do loop. The variable STATION_NUMBER is the counter for the iterative do statement which is also used as an index to the array of RSD structures. The primary station issues one command and receives one response from every secondary station each time through the loop. The first statement in the loop loads the secondary station address, indexed by STATION_NUMBER into the array of the RSD structures. Now when the primary sends a command, it will have the secondary’s address in the address field of the frame. The automatic address recognition feature is used by the primary to recognize the response from the secondary.

Next the main program determines the secondary stations state. Based on this state, the primary knows what command to send. If the station is in the DISCONNECT_S, the primary calls the SNRM_P

---

Figure 19-16. Remote Station Database Structure
procedure to try and put the secondary in the I_T_S. If the station state is in the GO_TO_DISC state, the DISC_P is called to try and put the secondary in the L_D_S. If the secondary is in neither one of the above two states, then it is in the I_T_S. When the secondary is in the I_T_S, the primary could send one of three commands: I, RR, or RNR. If the RSD’s buffer has data in it, indicated by INFO_LENGTH being greater than zero, and the secondary’s BUFFER_STATUS equals BUFFER_READY, then an I frame will be sent. Else if RPB=0, a RR supervisory frame will be sent. If neither one of these cases is true, then an RNR will be sent. The last statement in the main program checks the RPB bit. If set to one, the BUFFER_TRANSFER procedure is called, which transfers the data from the SIU receive buffer to the appropriate RSD buffer.

Receive Time Out

Each time a frame is transmitted, the primary sets a receive time out timer; Timer 0. If a response is not received within a certain time, the primary returns to the main program and continues polling the rest of the stations. The minimum length of time the primary should wait for a response can be calculated as the sum of the following parameters.

1. propagation time to the secondary station
2. clear-to-send time at the secondary station’s DCE
3. appropriate time for secondary station processing
4. propagation time from the secondary station
5. maximum frame length time

The clear-to-send time and the propagation time are negligible for a local network at low bit rates. However, the turnaround time and the maximum frame length time are significant factors. Using the 8044 secondaries in the AUTO mode minimizes turnaround time. The maximum frame length time comes from the fact the 8044 does not generate an interrupt from a received frame until it has been completely received, and the CRC is verified as correct. This means that the time-out is bit rate dependent.

Ns and Nr check Procedures

Each time an I frame or supervisory frame is received, the Nr field in the control byte must be checked. Since this data link only allows one outstanding frame, a valid Nr would satisfy either one of two equations; Ns(P) + 1 = Nr(S) the I frame previously sent by the primary is acknowledged, Ns(P) = Nr(S) the I frame previously sent is not acknowledged. If either one of these two cases is true, the CHECK_NR procedure returns a parameter of TRUE; otherwise a FALSE parameter is returned. If an acknowledgement is received, the Ns byte in the RSD structure is incremented, and the Information buffer may be cleared. Otherwise the information buffer remains full.

Figure 19-17. Block Diagram of Primary Station Software Structure
When an I frame is received, the Ns field has to be checked also. If Nr(P) = Ns(S), then the procedure returns TRUE, otherwise a FALSE is returned.

Receive Procedure

The receive procedure is called when a supervisory or information frame is sent, and a response is received before the time-out period. The RECEIVE procedure can be broken down into three parts. The first part is entered if an I frame is received. When an I frame is received, Ns, Nr and buffer overrun are checked. If there is a buffer overrun, or there is an error in either Ns or Nr, then the station state is set to GO_TO_DISC. Otherwise Nr in the RSD is incremented, the receive field length is saved, and the RPB bit is set. By incrementing the Nr field, the I frame just received is acknowledged the next time the primary polls the secondary with an I frame or a supervisory frame. Setting RBP protects the received data, and also tells the main program that there is data to transfer to one of the RSD buffers.

If a supervisory frame is received, the Nr field is checked. If a FALSE is returned, then the station state is set to GO_TO_DISC. If the supervisory frame received was an RNR, buffer status is set to not ready. If the response is not an I frame, nor a supervisory frame, then it must be an Unnumbered frame.

The only Unnumbered frames the primary recognizes are UA, DM, and FRMR. In any event, the station state is set to GO_TO_DISC. However if the frame received is a FRMR, Nr in the second data byte of the I field is checked to see if the secondary acknowledged an I frame received before it went into the FRMR state. If this is not done and the secondary acknowledged an I frame which the primary did not recognize, the primary transmits, the I frame when the secondary returns to the I_T_S. In this case, the secondary would receive duplicate I frames.
APPENDIX A
8044 SOFTWARE FLOWCHARTS
Figure 19-18. Secondary Station Driver Flow Chart
XMIT-UNNUMBERED PROCEDURE

TCB = CONTROL-BYTE

TBF = 1

RTS = 1

SI = 1

N

SI = 0

RETURN

TRANSMIT PROCEDURE

USER-STATE = CLOSED-S

N

STATUS = USER-STATE-CLOSE

STATION STATE = DISCONNECT_S

N

STATUS = LINK_DISCONNECTED

XMIT-BUFFER-LENGTH > BUFFER-LENGTH

N

XMIT-BUFFER-EMPTY = 0

TBL = XMIT-BUFFER-LENGTH

I-FRAME-LENGTH = XMIT-BUFFER-LENGTH

TBF = 1

STATUS = DATA-TRANSMITTED

RETURN STATUS

Figure 19-19. Secondary Station Driver Flow Chart
XMIT-FMR procedure

1. TCB = FRMR
2. TBS = FRMR-BUFFER (0)
3. TBL = 3
4. FRMR-BUFFER (0) = RCB
5. FRMR-BUFFERED (1) = NR NS
6. FRMR-BUFFER (2) = REASON
7. STATION-STATE = FRMR-S

8. \( P/F = 1 \)
   - N
   - Y

9. SEND FRMR FRAME
10. RETURN

Figure 19-20. Secondary Station Driver Flow Chart
IN-DISCONNECT-STATE PROCEDURE

USER-STATE = OPEN AND RCS = SNRM

CALL SNRM-RESPONSE

RETURN

SNRM-RESPONSE PROCEDURE

STATION-STATE = I-T-S

NS NR = 0

P/F = 1

CALL XMIT-UNNUMBERED (UA)

XMIT-BUFFER-EMPTY = 0

TBL = I-FRAME-LENGTH

TBF = 1

AM = 1

RETURN

Figure 19-21. Secondary Station Driver Flow Chart
Figure 19-22. Secondary Station Driver Flow Chart
Figure 19-23. Secondary Station Driver Flow Chart
Figure 19-25. Application Module Flow Chart
Figure 19-26. Application Module Flow Chart
MENU PROCEDURE

CALL OUTPUT-MESSAGE 'ENTER THE STATION ADDRESS:__'

CALL GET-HEX SHIFT TO LEFT BY FOUR

CALL GET-HEX OR WITH PREVIOUS HEX CHAR

LOAD ADDRESS INTO STAD

CALL OUTPUT-MESSAGE 'THE NEW STATION ADDRESS:__'

N

I = 2

CALL OPEN

RETURN

N

I = 3

CALL CLOSE

RETURN

RETURN

Y

I = 0

Y

I = 1

CALL OUTPUT-MESSAGE 'ENTER THE DESTINATION ADDRESS:__'

CALL GET-HEX SHIFT TO LEFT BY FOUR

CALL GET-HEX OR WITH PREVIOUS HEX CHAR

LOAD ADDRESS INTO DESTINATION-ADDRESS

CALL OUTPUT-MESSAGE 'THE NEW DESTINATION ADDRESS IS:__'

RETURN

N

RETURN

Figure 19-27. Application Module Flow Chart
ERROR PROCEDURE

Y

STATUS = PARITY ERROR

N

STATUS = OVER-RUN ERROR

ERR-MESSAGE-PTR = . PARITY

ERR-MESSAGE-PTR = . OVER-RUN

ERR-MESSAGE-PTR = . FRAMING

RESET ERROR FLAGS ON USART

ERR-MESSAGE-INDEX = 0

ERROR_FLAG = 1

TURN ON TX INTERRUPT

RETURN

Figure 19-28 Application Module Flow Chart
FIFO-T-OUT PROCEDURE

REMOVE CHAR FROM FIFO-T

IS FIFO-T EMPTY

Y

BUFFER_STATUS_T FULL

N

ARE THERE 80 BYTES LEFT IN FIFO

N

CLEAR TO SEND

BUFFER_STATUS_T = IN_USE

Y

CHAR = "LF" AND SEND-DATA > 0

N

DISABLE INTERRUPTS

BUFFER_STATUS_T = EMPTY

SEND-DATA = 0

ENABLE INTERRUPTS

SEND-DATA = SEND-DATA - 1

RETURN CHAR

Figure 19-29. Application Module Flow Chart
FIFO-T-IN PROCEEDURE

1. INSERT CHAR INTO FIFO-T

2. CHAR = 'LF'
   - Y: SEND-DATA = SEND-DATA + 1
   - N: BUFFER-STATUS-T = EMPTY

3. BUFFER-STATUS-T = EMPTY
   - Y: BUFFER-STATUS-T = IN USE
   - N: ARE THERE ONLY 20 BYTES AVAILABLE IN FIFO

4. ARE THERE ONLY 20 BYTES AVAILABLE IN FIFO
   - Y: NOT CLEAR TO SEND
     - Y: BUFFER-STATUS-T = FULL
       - Y: SEND-DATA = 0
         - Y: SEND-DATA = 1
         - N: RETURN
   - N: RETURN

Figure 19-30. Application Module Flow Chart
Figure 19-31. Application Module Flow Chart
POWER ON

INITIALIZE SIU REGISTERS

FOR EACH STATION
INITIALIZE RSD RECORDS
1. STATION-ADDRESS
2. STATION-STATE = DISCONNECT
3. BUFFER-STATE = BUFFER-NOT-READY
   4. INFO-LENGTH = 0

RETURN

Figure 19-32. Primary Station Flow Charts
Figure 19-33. Primary Station Flow Charts
Figure 19-34. Primary Station Flow Charts
Figure 19-35. Primary Station Flow Charts
BUFFER-TRANSFER PROCEDURE

DESTINATION ADDRESS FOUND IN RSD

N

RBP = 0

Y

BUFFER FULL IN RSD

N

MOVE DATA FROM SIU-RECV-BUFFER TO RSD BUFFER

RBP = 0

RETURN

Figure 19-36. Primary Station Flow Charts
Figure 19-37. Primary Station Flow Charts
Figure 19-38. Primary Station Flow Charts

- Remote Buffer Status = Buffer-Ready
- Remote Buffer Status = Buffer-Not-Ready
- Station-State = Go-To-Disc
- Station-State = Disconnect-S
- Remote Buffer Status: Go-To-Disc
- Remote Buffer Status: FRMR
- Remote Buffer Status: RNR
- Remote Buffer Status: I Frame
- Remote Buffer Status: Supervisory Frame
- Remote Buffer Status: N Frame
- Remote Buffer Status: Acknowledged Frame
- Remote Buffer Status: Unacknowledged Frame
- Remote Buffer Status: No Frame
- Remote Buffer Status: Error Frame
- Remote Buffer Status: Unknown Frame
- Remote Buffer Status: Ignore Frame
- Remote Buffer Status: Stop Frame
- Remote Buffer Status: Go-To-Disc
- Remote Buffer Status: FRMR
- Remote Buffer Status: RNR
- Remote Buffer Status: I Frame
- Remote Buffer Status: Supervisory Frame
- Remote Buffer Status: N Frame
- Remote Buffer Status: Acknowledged Frame
- Remote Buffer Status: Unacknowledged Frame
- Remote Buffer Status: No Frame
- Remote Buffer Status: Error Frame
- Remote Buffer Status: Unknown Frame
- Remote Buffer Status: Ignore Frame
- Remote Buffer Status: Stop Frame
- Remote Buffer Status: Go-To-Disc
- Remote Buffer Status: FRMR
- Remote Buffer Status: RNR
- Remote Buffer Status: I Frame
- Remote Buffer Status: Supervisory Frame
- Remote Buffer Status: N Frame
- Remote Buffer Status: Acknowledged Frame
- Remote Buffer Status: Unacknowledged Frame
- Remote Buffer Status: No Frame
- Remote Buffer Status: Error Frame
- Remote Buffer Status: Unknown Frame
- Remote Buffer Status: Ignore Frame
- Remote Buffer Status: Stop Frame
- Remote Buffer Status: Go-To-Disc
- Remote Buffer Status: FRMR
- Remote Buffer Status: RNR
- Remote Buffer Status: I Frame
- Remote Buffer Status: Supervisory Frame
- Remote Buffer Status: N Frame
- Remote Buffer Status: Acknowledged Frame
- Remote Buffer Status: Unacknowledged Frame
- Remote Buffer Status: No Frame
- Remote Buffer Status: Error Frame
- Remote Buffer Status: Unknown Frame
- Remote Buffer Status: Ignore Frame
- Remote Buffer Status: Stop Frame
- Remote Buffer Status: Go-To-Disc
- Remote Buffer Status: FRMR
- Remote Buffer Status: RNR
- Remote Buffer Status: I Frame
- Remote Buffer Status: Supervisory Frame
- Remote Buffer Status: N Frame
- Remote Buffer Status: Acknowledged Frame
- Remote Buffer Status: Unacknowledged Frame
- Remote Buffer Status: No Frame
- Remote Buffer Status: Error Frame
- Remote Buffer Status: Unknown Frame
- Remote Buffer Status: Ignore Frame
- Remote Buffer Status: Stop Frame
- Remote Buffer Status: Go-To-Disc
- Remote Buffer Status: FRMR
- Remote Buffer Status: RNR
- Remote Buffer Status: I Frame
- Remote Buffer Status: Supervisory Frame
- Remote Buffer Status: N Frame
- Remote Buffer Status: Acknowledged Frame
- Remote Buffer Status: Unacknowledged Frame
- Remote Buffer Status: No Frame
- Remote Buffer Status: Error Frame
- Remote Buffer Status: Unknown Frame
- Remote Buffer Status: Ignore Frame
- Remote Buffer Status: Stop Frame
APPENDIX B
LISTINGS OF SOFTWARE MODULES
PL/M-51 COMPILER  RUPI-44 Secondary Station Driver

ISIS-II PL/M-51 V1.0
COMPILER INVOKED BY:  .F2 PLM51 :F2.APMOTE.SRC

$TITLE  ('RUPI-44 Secondary Station Driver')
@DEBUG
@REGISTERBANK(1)
MAINPROG.B10
@NOLIST

/* To save paper the RUPI registers are not listed, but this is the statement
used to include them $INCLUDE (.F2 REQ44.DCL) */

5 1 DECLARE LIT LITERALLY 'LITERALLY',
    TRUE   LIT '0FFH',
    FALSE  LIT '00H',
    FOREVER LIT 'WHILE 1';

/* SDLc commands and responses */

6 1 DECLARE SMRM LIT '83H',
    UA     LIT '73H',
    DISC   LIT '43H',
    DM     LIT '1FH',
    FRMR   LIT '97H',
    RES_DISC LIT '53H',
    UP     LIT '33H',
    TEST   LIT '0EH'.

/* User states */
OPEN_S   LIT '00H',
CLOSED_S LIT '01H',

/* Station states */
DISCONNECT_S LIT '00H',  /* LOGICALLY DISCONNECTED STATE */
FRMR_S   LIT '01H',  /* FRAME REJECT STATE */
I_T_S    LIT '02H',  /* INFORMATION TRANSFER STATE */

/* Status values returned from TRANSMIT procedure */
USER_STATE_CLOSED LIT '00H',
LINK_DISCONNECTED LIT '01H',
OVERFLOW    LIT '02H',
DATA_TRANSMITTED LIT '03H'.

/* Parameters passed to XMIT_FRMR */
UNABBINDED_C LIT '00H',
NO_1_FIELD_ALLOWED LIT '01H',
BUFF_OVERUN LIT '02H',
SES_ERR LIT '03H'.

19-53
PL/I-51 Compiler  RUP1-44 Secondary Station Driver

/* Variables */
USER_STATE BYTE AUXILIARY,
STATION_STATE BYTE AUXILIARY,
I_FRAME_LENGTH BYTE AUXILIARY.

/* Buffers */
BUFFER_LENGTH LIT '60',
SIU_XMIT_BUFFER(BUFFER_LENGTH) BYTE PUBLIC IDATA,
SIU_RECV_BUFFER(BUFFER_LENGTH) BYTE PUBLIC,
FROM_BUFFER(3) BYTE.

/* Flags */
XMIT_BUFFER_EMPTY BIT PUBLIC;

7 2 SIU_RECV: PROCEDURE (LENGTH) EXTERNAL;
8 2 DECLARE LENGTH BYTE;
9 1 END SIU_RECV;

10 2 OPEN: PROCEDURE PUBLIC USING 2;
11 2 USER_STATE=OPEN_S;
12 1 END OPEN;

13 2 CLOSE: PROCEDURE PUBLIC USING 2;
14 2 AM=0;
15 2 USER_STATE=CLOSED_S;
16 1 END CLOSE;

17 2 POWER_ON_D: PROCEDURE PUBLIC USING 0;
18 2 USER_STATE=CLOSED_S;
19 2 STATION_STATE=DISCONNECT_S;
20 2 TBG= SIU_XMIT_BUFFER(0);
21 2 RBB= SIU_RECV_BUFFER(0);
22 2 RBL=BUFFER_LENGTH;
23 2 RBE=1; /* Enable the SIU's receiver */
24 2 XMIT_BUFFER_EMPTY=1;
25 1 END POWER_ON_D;

26 2 TRANSMIT: PROCEDURE (XMIT_BUFFER_LENGTH) BYTE PUBLIC USING 0;
/* User must check XMIT_BUFFER_EMPTY flag before calling this procedure */

27 2 DECLARE XMIT_BUFFER_LENGTH BYTE,
I BYTE AUXILIARY,
STATUS BYTE AUXILIARY;

28 2 IF USER_STATE=CLOSED_S
THEN STATUS=USER_STATE_CLOSED;

30 2 ELSE IF STATION_STATE=DISCONNECT_S
THEN STATUS=LINK_DISCONNECTED;

32 2 ELSE IF XMIT_BUFFER_LENGTH>BUFFER_LENGTH
THEN STATUS=OVERFLOW;

34 3 ELSE DO;

19-54
DECLARE CONTROL_BYTE BYTE;

TCB=CONTROL_BYTE;

TBF=1;

RTB=1;

DO WHILE NOT SI;

END;

SI=0;

DECLARE XMIT_UNNUMBERED: PROCEDURE (CONTROL_BYTE);

DECLARE SNRM_RESPONSE: PROCEDURE;

DECLARE XMIT_FRMR: PROCEDURE (REASON);

DECLARE FRMR_BUFFER: BYTES;

DECLARE RC8: BYTE;

DECLARE REASON BYTE;

TCB=FRMR;

TB=FRMR_BUFFER(0);

TB=3;

FRMR_BUFFER(0)=RC8;

FRMR_BUFFER(1)=(SHL((NSBR AND OEH),4) OR SHR((NSBR AND OEHENH),4));

DO CASE REASON;

FRMR_BUFFER(2)=01H; /* UNASSIGNED_C */
```plaintext
76 3 FRMR_BUFFER(2)=02H; /* NO_I_FIELD_ALLOWED */
77 3 FRMR_BUFFER(2)=04H; /* BUFF_OVERFLOW */
78 3 FRMR_BUFFER(2)=08H; /* BGR_ERR */
79 3 END;
80 2 STATION_STATE=FRMR_SI;
81 2 IF (RCB AND 10H) <> 0 THEN DO;
83 3 TBF=1;
84 3 RTS=1;
85 4 DO WHILE(NOT SI);
86 4 END;
87 3 SI=0;
88 3 END;
89 1 END XMIT_FRMR;
90 2 IN_DISCONNECT_STATE: PROCEDURE /* Called from SIU_INT procedure */
91 2 IF ((UBER_STATE=OPEN_SI) AND (RCB AND OEFH)) SNRM) THEN CALL SNRM_RESPONSE;
93 2 ELSE IF (RCB AND 10H) <> 0 THEN DO;
95 3 TBL=0;
96 3 CALL XMIT_UNNUMBERED(DM);
97 3 END;
98 1 END IN_DISCONNECT_STATE;
99 2 IN_FRMR_STATE: PROCEDURE /* Called by SIU_INT when a frame has been received when in the FRMR state */
100 2 IF (RCB AND OEFH) SNRM THEN DO;
102 3 CALL SNRM_RESPONSE;
103 3 TBS= SIU_XMIT_BUFFER(0); /* Restore transmit buffer start address */
104 3 END;
105 2 ELSE IF (RCB AND OEFH) DISC THEN DO;
107 3 STATION_STATE=DfnCONNECT_SI;
108 3 TBS= SIU_XMIT_BUFFER(0); /* Restore transmit buffer start address */
109 3 IF (RCB AND 10H) <> 0 THEN DO;
111 4 TBL=0;
112 4 CALL XMIT_UNNUMBERED(UA);
113 4 END;
114 3 END;
115 3 ELSE DO. /* Receive control byte is something other than DISC or SNRM */
116 3 IF (RCB AND 10H) <> 0 THEN DO;
118 4 TBF=1;
119 4 RTS=1;

19-56
```
PL/M-51 COMPILER RUP1-44 Secondary Station Driver

120 3 DO WHILE NOT SI;
121 4 END;
122 3 END;
123 1 END IN_FRM_STATE;
124 2 COMMAND_DECODE: PROCEDURE .
125 2 IF (RCB AND OEFH)=SNRM
126 3 THEN CALL SNRM_RESPONSE;
127 2 ELSE IF (RCB AND OEFH)=DISC
128 3 THEN DO;
129 4 STATION_STATE=DISCONNECT_S;
130 3 IF (RCB AND 1OH)<>0
131 4 THEN DO;
132 5 TBL=0;
133 4 CALL XMIT_UNNUMBERED(UA);
134 4 END;
135 3 END;
136 2 ELSE IF (RCB AND OEFH)=TEST
137 2 THEN DO;
138 3 IF (RCB AND 1OH)>0 /* Respond if polled */
139 4 THEN DO; /* FOR BDV=1, SEND THE TEST RESPONSE WITHOUT AN I FIELD */
140 5 IF (BDV=1)
141 6 THEN DO;
142 7 TBL=0;
143 6 CALL XMIT_UNNUMBERED(TEST OR 1OH);
144 5 END;
145 6 ELSE DO; /* IF no BDV, send received I field back to primary */
146 7 TBL=RF;
147 6 CALL XMIT_UNNUMBERED(TEST OR 1OH);
148 5 TBS=RBS;
149 6 END;
150 5 CALL XMIT_UNNUMBERED(1TEST OR 1OH);
151 5 END;
152 4 ELSE IF (RCB AND 1OH)=0 /* Kicked out of the AUTO mode because an I frame was received while RPB = 1 */
153 5 THEN DO;
154 6 IF XMIT_BUFFER_EMPTY=O
155 7 THEN DO;
156 8 TBL=I_FRAME_LENGTH,
157 7 TBF=1;
158 6 END;
159 5 AM=1;
160 3 END;
161 2 ELSE IF (RCB AND O1H) = O /* Kicked out of the AUTO mode because */
162 3 THEN DO;
163 3 IF XMIT_BUFFER_EMPTY = 1
164 4 THEN TBL=0;
165 3 TBF=1. /* Send an AUTO mode response */
DECLARE I BYTE AUXILIARY;

SI=0;

IF STATION_STATE<>I_T_S /* Must be in NON-AUTO mode */
    THEN DO:
        IF RBE=0 /* Received a frame? Give response */
            THEN DO:
                /* DO CASE STATION_STATE; */
                CALL IN_DISCONNECT_STATE;
                CALL IN_FRMR_STATE;
                END;
            ELSE IF RBE<>1
                THEN RETURN;
        END;
    ELSE DO CASE STATION_STATE,
        CALL IN_PIS_CONNECT_STATE,
        CALL IN_FRMR_STATE;
        END.

RETURN;

RBE=1.

ELSE IF RBF=1
    THEN XMIT_BUFFER_EMPTY=1. /* TRANSMITTED A FRAME */
    ELSE RBF<>0
        THEN DO;

Software and application note written by Charles Yager

WARNING:
4 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>CODE SIZE</th>
<th>CONSTANT SIZE</th>
<th>DIRECT VARIABLE SIZE</th>
<th>INDIRECT VARIABLE SIZE</th>
<th>BIT SIZE</th>
<th>BIT-ADDRESSABLE SIZE</th>
<th>AUXILIARY VARIABLE SIZE</th>
<th>MAXIMUM STACK SIZE</th>
<th>REGISTER-BANK(S) USED</th>
<th>460 LINES READ</th>
<th>0 PROGRAM ERROR(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>02BFH</td>
<td>0000H</td>
<td>3FH+02H</td>
<td>0H+00H</td>
<td>0000H</td>
<td>0017H</td>
<td>012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TITLE ('Application Module Async/SDLC Protocol converter')
\*debug
\*registerbank(0)
user
mod:do;
\*ENDLIST
5 1 DECLARE LIT LITERALLY 'LITERALLY', TRUE LIT 'OFFH', FALSE LIT 'O0H', FOREVER LIT 'WHILE 1', ESC LIT '1BH', LF LIT '0AH', CR LIT '0DH', BS LIT '0BH', BEL LIT '07H', EMPTY LIT '00H', INUSE LIT '01H', FULL LIT '02H', USER_STATE_CLOSED LIT '03H', LINK_DISCONNECTED LIT '01H', OVERFLOW LIT '02H', DATA_TRANSMITTED LIT '03H'.
BUFFERS
BUFFER_LENGTH LIT '40', SIU_XMIT_BUFFER(BUFFER_LENGTH) BYTE EXTERNAL IDATA, SIU_RECV_BUFFER(BUFFER_LENGTH) BYTE EXTERNAL, FIFO_T(1256) BYTE AUXILIARY, IN_PTR_T BYTE AUXILIARY, OUT_PTR_T BYTE AUXILIARY, BUFFER_STATUS_T BYTE AUXILIARY, FIFO_R(1256) BYTE AUXILIARY, IN_PTR_R BYTE AUXILIARY, OUT_PTR_R BYTE AUXILIARY, BUFFER_STATUS_R BYTE AUXILIARY.

Variables and Parameters
LENTH BYTE AUXILIARY, CHAR BYTE AUXILIARY, I BYTE AUXILIARY, USART_CMD BYTE AUXILIARY, DESTINATION_ADDRESS BYTE AUXILIARY, SEND_DATA BYTE AUXILIARY, RESULT BYTE AUXILIARY, ERR_MESSAGE_INDEX BYTE AUXILIARY, ERR_MESSAGE_PTR WORD AUXILIARY,
Messages Sent to the Terminal
PARITY(*) BYTE CONSTANT(LF.CR. 'Parity Error Detected';LF.CR.OOH), FRAME(*) BYTE CONSTANT(LF.CR. 'Framing Error Detected';LF.CR.OOH).
OVER_RU8N(*) BYTE CONSTANT(LF, CR, 'Overrun Error Detected': LF, CR, 0).
LINK(*) BYTE CONSTANT(LF, CR, 'Unable to Get Online': LF, CR, 0).
DEST_ADDR(*) BYTE CONSTANT(CR, LF, LF, 'Enter the destination address: ___ ', BS, BS, 0).
D_ADDR_ACK(*) BYTE CONSTANT(CR, LF, LF, 'The new destination address is ': CR, 0).
STAT_ADDR(*) BYTE CONSTANT(CR, LF, LF, 'Enter the station address: ___ ', BS, BS, 0).
S_ADDR_ACK(*) BYTE CONSTANT(CR, LF, LF, 'The new station address is ': CR, 0).
ADDR_ACK_FIN(*) BYTE CONSTANT('H', CR, LF, LF, 0).

SIGN.ON(*) BYTE CONSTANT(CR, LF, LF, '(()/RUP-44 Secondary Station': CR, LF, LF, '1 - Set the Station Address': LF, CR,
'2 - Set the Destination Address': CR, LF, '3 - Go Online': CR, LF, '4 - Go Offline': CR, LF, '5 - Return to terminal mode': CR, LF, LF, 'Enter option: ___ ', BS, BS, 0).
FIN(*) BYTE CONSTANT(CR, LF, LF, 0);

/* Characters Received From the Terminal */
HEX_TABLE(17) BYTE CONSTANT('0123456789ABCDEF', BEL),
MENU_CHAR(6) BYTE CONSTANT('12345', BEL),

/* Flags and Bits */
XMIT_BUFFER EMPTY BIT EXTERNAL, /* Semaphore for RUPI SIOU Transmit Buffer */
STOP BIT BIT AT(147) REG, /* Terminal parameters */
ECHO BIT AT(084H) REG,
WAIT BIT, /* Timeout flag */
ERROR_FLAG BIT, /* Error message Flag */

/* Peripheral Addresses */
USART STATUS BYTE AT(OBO1H) AUXILIARY,
USART DATA BYTE AT(OBOOH) AUXILIARY,
TIMER 0 BYTE AT(OO0OH) AUXILIARY,
TIMER 1 BYTE AT(1001H) AUXILIARY,
TIMER 2 BYTE AT(1002H) AUXILIARY,

/* External Procedures */
```plaintext
6 2  POWER_ON_D:  PROCEDURE  EXTERNAL;
7   1  END  POWER_ON_D;
8 2  CLOSE:  PROCEDURE  EXTERNAL  USING  2;
9   1  END  CLOSE;
10 2  OPEN:  PROCEDURE  EXTERNAL  USING  2;
11 1  END  OPEN;
12 2  TRANSMIT:  PROCEDURE  (XMIT_BUFFER_LENGTH)  BYTE  EXTERNAL;
13 2  DECLARE  XMIT_BUFFER_LENGTH  BYTE;
14 1  END  TRANSMIT;

/* Local Procedures */
15 2  TIMER_0_INT:  PROCEDURE  INTERRUPT  1  USING  1;
16 2  WAIT=0,
17 1  END  TIMER_0_INT;
18 2  POWER_ON:  PROCEDURE  USING  0;
19 2  DECLARE  TEMP  BYTE  AUXILIARY;
20 2  SMD=94H;  /* Using DPLL, NRZI, PBS, TIMER 1, @ 62.5 Kbps */
21 2  TMOD=21H;  /* Timer 0 16 bit, Timer 1 auto reload */
22 2  TH1=OFFH;
23 2  TCON=40H;
24 2  TIMER_CONTROL=37H;  /* Initialize USART's system clock: 8254 */
25 2  TIMER_0=04H;
26 2  TIMER_O=00H;
27 2  TIMER_CONTROL=77H;  /* Initialize TxC, RxC */

/* Definition for dip switch tied to P1.0 to P1.6 */
Bit Rate  3  2  1
---  ---  ---  ---
300  on  on  on
1200 on on off
2400 on off on
4800 on off off
9600 off on on
19200 off on off

Stop bit 4
---
1 on
2 off

Parity 6  5
---  ---
off on on
even off on
odd off off
```
Echo on

on

off

off

/* Read the dip switch to determine the bit rate */

28 2 TEMP=P1 AND 07H;  

29 2 IF TEMP>0;  

30 2 THEN TEMP=0;  

31 3 DO CASE TEMP;  

32 4 /* 300 */  

33 4 DO;  

34 4 TIMER_1=03H;  

35 4 TIMER_1=02H;  

36 4 END;  

37 4 /  

38 4 TIMER_1=02H;  

39 4 TIMER_1=05H;  

40 4 /  

41 4 TIMER_1=03H;  

42 4 END;  

43 4 /* 1200 */  

44 4 DO;  

45 4 TIMER_1=02H;  

46 4 END;  

47 4 /* 2400 */  

48 4 DO;  

49 4 TIMER_1=01H;  

50 4 END;  

51 4 /* 4800 */  

52 4 DO;  

53 4 TIMER_1=00H;  

54 4 END;  

55 4 /* 9600 */  

56 4 DO;  

57 4 TIMER_1=33H;  

58 4 END;  

59 4 /* 19200 */  

60 4 DO;  

61 4 TEMP=TEMP OR (P1 AND 30H);  

62 4 IF STOP_BIT=1  

63 4 THEN TEMP=TEMP OR 00H;  

64 4 ELSE TEMP=TEMP OR 40H;  

65 2 /  

66 2 USART_STATUS=TEMP;  

67 2 /  

68 2 STAD=OFFH;
DECLISTإقت/(:0; I# Intiizci Eags #/
IN_PTR.T. OUT_PTR.T, IN_PTR_R, OUT_PTR_R = 0; /* Initialize FIFO PTRs*/
BUFFER_STATUS.T. BUFFER_STATUS_R= EMPTY;
CALL POWER_ON;D;
IF=01H; /* USART's Rsdy is the highest priority */
IE=93H; /* Enable USART Rsdy, SI. and Timer 0 interrupts*/
ERROR_FLAG=0;
END POWER_ON;
FIFO_R_IN:Procedure (CHAR) USING I;
DECLARE CHAR BYTE;
FIFO_R(IN_PTR_R)=CHAR;
IN_PTR_R=IN_PTR_R+l;
IF BUFFER_STATUS_R=EMPTY
THEN DO;
ELSE IF ((BUFFER_STATUS_R=INUSE) AND (IN_PTR_R=OUT_PTR_R))
THEN BUFFER_STATUS_R=FULL;
END FIFO_R_IN;
FIFO_R_OUT: Procedure BYTE USING I;
DECLARE CHAR BYTE AUXILIARY;
CHAR=FIFO_R(OUT_PTR_R),
OUT_PTR_R=OUT_PTR_R+1;
IF OUT_PTR_R=IN_PTR_R
THEN DO;
ELSE IF ((BUFFER_STATUS_R=FULL) AND (OUT_PTR_R-2=IN_PTR_R))
THEN BUFFER_STATUS_R=INUSE;
END;
RETURN CHAR;
END FIFO_R_OUT;
USART_XMIT_INT: Procedure INTERRUPT 2 USING I;
DECLARE MESSAGE BASED ERR_MESSAGE_PTR(1) BYTE CONSTANT.

IF ERROR_FLAG THEN DO:
   IF MESSAGE(ERR_MESSAGE_INDEX)<>0 /* Then continue to send the message */
   THEN DO:
      USART_DATA = MESSAGE(ERR_MESSAGE_INDEX);
      ERR_MESSAGE_INDEX = ERR_MESSAGE_INDEX + 1;
      END;
   ELSE DO: /* If message is done reset ERROR_FLAG and shut off interrupt if FIFO is empty */
      ERROR_FLAG = 0;
      IF BUFFER_STATUS_R= EMPTY THEN ERR = 0;
      END;
   END;
ELSE USART_DATA = FIFO_R_OUT;

END USART_XMIT_INT:

SIU_RECV: PROCEDURE (LENGTH) PUBLIC USING 1.

DECLARE LENGTH BYTE.
1 BYTE AUXILIARY;
DO 1=0 TO LENGTH-1;
DO WHILE BUFFER_STATUS_R=FULL; /* Check to see if fifo is full */
END;
CALL FIFO_R_IN(SIU_RECV_BUFFER(I));
END;

FIFO_T_IN: PROCEDURE (CHAR) USING 2;

DECLARE CHAR BYTE;
IFCHAR=LF THEN SEND_DATA=SEND_DATA+1;
ELSE IF BUFFER_STATUS_T=EMPTY THEN BUFFER_STATUS_T=INUSE;
ELSE IF ((BUFFER_STATUS_T=INUSE) AND (IN_PTR_T+20=OUT_PTR_T))
   THEN DO; /* Stop reception using CTS */
      USART_STATUS, USART_CMD=USART_CMD AND NOT(2OH);
      BUFFER_STATUS_T=FULL;
      IF SEND_DATA=0 THEN SEND_DATA=1; /* If the buffer is full and no LF has been received then send data */
   END;
END FIFO_T_IN:
143 2 FIFO_T_OUT:  PROCEDURE BYTE ;
146 2 DECLARE  CHAR  BYTE  AUXILIARY ;
147 2 CHAR=FIFO_T(OUT_PTR_T); 
148 2 OUT_PTR_T=OUT_PTR_T+1;
149 2 IF OUT_PTR_T=IN_PTR_T  /* Then FIFO_T is empty */ 
150 2 THEN  DO; 
151 3 EA=0; 
152 3 BUFFER_STATUS_T=EMPTY; 
153 3 SEND_DATA=0; 
154 3 EA=1; 
155 3 END; 
156 2 ELSE IF ((BUFFER_STATUS_T=FULL) AND (OUT_PTR_T=IN_PTR_T))
157 2 THEN DO; 
158 3 USART_STATUS=USART_CMD=USART_CMD OR 20H.
159 3 BUFFER_STATUS_T=INUSE; 
160 3 END; 
161 2 IF (CHAR=LF AND SEND_DATA) THEN SEND_DATA=SEND_DATA+1; 
162 2 RETURN CHAR; 
164 1 END FIFO_T_OUT;

165 2 ERROR:  PROCEDURE (STATUS) USING 2;
166 2 DECLARE  STATUS  BYTE ;
167 2 IF (STATUS AND 08H)=0  
168 2 THEN  ERR_MESSAGE_PTR=PARITY; 
169 2 ELSE IF (STATUS AND 10H)=0  
170 2 THEN  ERR_MESSAGE_PTR=OVER_RUN; 
171 2 ELSE IF (STATUS AND 20H)=0  
172 2 THEN  ERR_MESSAGE_PTR=FRAME; 
173 2 USART_STATUS=USART_CMD OR 10H); /* Reset error flags on USART */
174 2 ERR_MESSAGE_INDEX = 0;
175 2 ERROR_FLAG=1;  /* Turn on T6 Interrupt */
176 2 EX1=1; 
177 1 END ERROR;

178 2 LINK_DISC:  PROCEDURE ;
/* This procedure sends the message 'Unable to Get Online' to the terminal */
179 2 DECLARE MESSAGE_PTR WORD AUXILIARY, 
180 2 MESSAGE BASED MESSAGE_PTR(1) BYTE CONSTANT, 
181 2 J EX1_STORE BIT; 
182 2 EX1_STORE=EX1;  /* Shut off async transmit interrupt */
183 2 EX1=0;
184 2 MESSAGE_PTR=LINK, 
185 2 J=0;
186 2 DO WHILE (MESSAGE(J)<>));
DO WHILE (USART_STATUS AND 01H) = 0; /* Wait for TXRDY on USART */
endl:

DO WHILE (USART_STATUS AND 01H) = 0;
end:

DO WHILE (USART_STATUS AND 02H) = 0;
end:

RETURN USART_DATA;
end GET_HEX:

DO I=0 TO 15:
end:

DO I=0 TO 15:
end:

RETURN I;
end GET_HEX:

DO WHILE MESSAGE[I] <> 0:
end:

DO WHILE MESSAGE[I] <> 0:
end:

I=I+1;
end:

I=I+1;
end:

# Restore async transmit interrupt */
END

# Restore async transmit interrupt */
END

END

END

END

END

END

END

END

END

END

END

END

END

END

END

END

END

END
DECLARE:

I BYTE AUXILIARY,
CHAR BYTE AUXILIARY,
STATION_ADDRESS BYTE AUXILIARY;

START:
CALL OUTPUT_MESSAGE(SIGN_ON);

DO I=0 TO 4;
   IF CHAR=MENU_CHAR(I)
      THEN GOTO M1;
END;

M1 CALL CO(MENU_CHAR(I));
IF I=9
   THEN GOTO MO;

DO CASE I,

   CALL OUTPUT_MESSAGE(STAT_ADDR);
   STATION_ADDRESS=SHL(GET_HEX.4),
   STATION_ADDRESS=(STATION_ADDRESS OR GET_HEX);
   STAD=STATION_ADDRESS;
   CALL OUTPUT_MESSAGE(S_ADDR_ACK),
   CALL CO(HEx_TABLE(BHR(STATION_ADDRESS.4))),
   CALL CO(HEx_TABLE(OFH AND STATION_ADDRESS)),
   CALL OUTPUT_MESSAGE(ADDR_ACK_FIN),
END;

DO,

   CALL OUTPUT_MESSAGE(DEST_ADDR),
   DESTINATION_ADDRESS=SHL(GET_HEX.4),
   DESTINATION_ADDRESS=(DESTINATION_ADDRESS OR GET_HEX),
   CALL OUTPUT_MESSAGE(D_ADDR_ACK).
CALL COCHEX_TABLE(CSHRS_HAND_DESTINATION_ADDRESS, 4),
CALL COCHEX_TABLE(COFS_HAND_DESTINATION_ADDRESS),
CALL OUTPUT_MESSAGE(ADDR_ACK_IN),
END.
DO.
CALL OUTPUT_MESSAGE(C)
END.
DO.
CALL OUTPUT_MESSAGE(FIN)
CALL CLOSE
END.
DO CASE */

USART_RECV_INT PROCEDURE INTERRUPT 0 USING 2:
DECLARE CHAR BYTE AUXILIARY.
STATUS BYTE AUXILIARY.
CHAR=USART_DATA;
STATUS=USART_STATUS AND 3BH,
IF STATUS<>O THEN CALL ERROR(STATUS);
ELSE IF CHAR=ESC THEN CALL MENU;
ELSE DO.
CALL FIFO_T_IN(CHAR),
IF ECHO=O THEN CALL CO(CHAR),
END.
END
BEGIN
CALL POWER_ON.
DO FOREVER.
IF SEND_DATA>O THEN DO.
DO WHILE NOT(XMIT_BUFFER_EMPTY); /*Wait until XMIT_BUFFER is empty*/
END.
LENGTH CHAR =1,
SIU_XMIT_BUFFER(0)=$=DESTINATION_ADDRESS;
DO WHILE ((CHAR<>LF) AND (LENGTH<BUFFER_LENGTH) AND (BUFFER_STATUS<>EMPTY));
CHAR=FIFO_T_OUT;
SIU_XMIT_BUFFER(LENGTH)=CHAR;
LENGTH=LENGTH+1;
END;

/* If the line entered at the terminal is greater than BUFFER_LENGTH char. send the
first BUFFER_LENGTH char. then send the rest; since the SIU buffer is only BUFFER_LENGTH bytes */
L1 =O. /* Use I to count the number of unsuccessful
transmits */

RETRY: RESULT=TRANSMIT(LENGTH); /* Send the message */
IF RESULT=DATA_TRANSMITTED
THEN DO:
/\ Wait 50 msec for link to connect then try again */
WAIT=1;
THO=0CH;
TLO=OAPH;
TR0=11;
DO WHILE WAIT;
END;
I=I+1;
IF I>100 THEN DO: /* Wait 5 sec to get on line else
send error message to terminal
and try again */
CALL LINK_DISC.
GO TO L1.
END.

END,
END,
END.
END.
END:
END
END USER.MOD.

WARNINGS
2 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION

| CODE SIZE | 0682H | 1714D |
| CONSTANT SIZE | 01CPH | 443D |
| DIRECT VARIABLE SIZE | 00H+05H | 0D+ 5D |
| INDIRECT VARIABLE SIZE | 00H+04H | 0D+ 0D |
| BIT SIZE | 02H+01H | 2D+ 1D |
| BIT-ADDRESSABLE SIZE | 00H+00H | 0D+ 0D |
| AUXILIARY VARIABLE SIZE | 021PH | 543D |
| MAXIMUM STACK SIZE | 002BH | 40D |
| REGISTER-BANK(S) USED | 0 1 2 |
| 713 LINES READ | 0 |
| 0 PROGRAM ERROR(S) |

END OF PL/M-81 COMPILATION
declare LIT LITERALLY 'LITERALLY', TRUE LIT 'OFFH', FALSE LIT '00H', FOREVER LIT 'WHILE 1'.

remote station buffer status */
BUFFERS READY LIT '0', BUFFER NOT READY LIT '1',

station states */
DISCONNECT S LIT '00H', /* LOGICALLY DISCONNECTED STATE*/ GO TO DISC LIT '01H', I T S LIT '02H', /* INFORMATION TRANSFER STATE*/

parameters passed to XMIT I T S */
T I FRAME LIT '00H', T RR LIT '01H', T RNR LIT '02H',

secondary station identification */
NUMBER OF STATIONS LIT '2', SECONDARY ADDRESSES (NUMBER OF STATIONS) BYTE CONSTANT ('92H, 43H'),
/* Remote Station Database */

RSD(NUMBER_OF_STATIONS) STRUCTURE
(STATION_ADDRESS BYTE,
 STATION_STATE BYTE,
 NS BYTE,
 NR BYTE,
 BUFFER_STATUS BYTE. /* The status of the secondary stations buffer */
 INFO_LENGTH BYTE,
 DATA(64) BYTE) AUXILIARY.

/* VARIABLES */

STATION_NUMBER BYTE AUXILIARY.
RECV_FIELD_LENGTH BYTE AUXILIARY.
WAIT BIT.

/* BUFFERS */

SIU_XMIT_BUFFER(64) BYTE IDATA.
SIU_RECV_BUFFER(64) BYTE.

POWER_ON: PROCEDURE:

DECLARE I BYTE AUXILIARY.

TBS= SIU_XMIT_BUFFER(0);
RBS= SIU_RECV_BUFFER(0);
RBL=64; /* 64 Byte receive buffer */
RBE=1; /* Enable the SIU's receiver */

DO I= 0 TO NUMBER_OF_STATIONS-1.

RSD(I).STATION_ADDRESS=SECONDARY_ADDRESSES(I).
RSD(I).STATION_STATE=DISCONNECT.
RSD(I).BUFFER_STATUS=BUFFER_NOT_READY.
RSD(I).INFO_LENGTH=0.

END:

END POWER_ON.

XMIT: PROCEDURE (CONTROL_BYTE).

DECLARE CONTROL_BYTE BYTE.

TCB=CONTROL_BYTE.

19-71
RTS=1,
DO WHILE NOT SI.
END.
SI=0.
END XMIT.

TIME_O_INT PROCEDURE INTERRUPT I USING 1.
WAIT=0.
END TIMER_O_INT.

TIME_OUT PROCEDURE BYTE.
/* Time_out returns true if there wasn’t
a frame received within 200 msec.
If there was a frame received within
200 msec then time_out returns false. */
DECLARE I BYTE AUXILIARY.
DO I=0 TO 3.
WAIT=1.
TH0=3CH.
TL0=0A8H.
TR0=1.
DO WHILE WAIT.
IF SI=1
THEN GOTO T_01.
END.
RETURN TRUE.
SI=0.
RETURN FALSE.
END TIME_OUT.
SEND_DISC PROCEDURE.
TBL=0.
CALL XMIT(DISC).
IF TIME_OUT=FALSE
THEN IF RC3=UP OR RC8=DM
THEN DO.
RSD(STATION_NUMBER) BUFFER_STATUS=BUFFER_NOT_READY,
RSD(STATION_NUMBER) STATION_STATE=DISCONNECTED.
END.
RBE=1.
END SEND_DISC.

SEND_NM_PROCEDURE.
TBL=0.
PL/M-51 COMPILER 204744 Primary Station

CALL XMIT(SNRM).

IF (TIME_OUT=FALSE) AND (RCS=UA) THEN DO.

RSD(STATION_NUMBER) STATION_STATE=I.T.S.
RSD(STATION_NUMBER) NS=0.
RSD(STATION_NUMBER) NR=0.
END.

RBE=1.

END SEND_SNRM.

CHECK_NS PROCEDURE BYTE.
/* Check the NS Field of the received frame If Nr(P)=Ns(S) return true */

IF (RSD(STATION_NUMBER) NR=(SHR(RCB,1) AND O7H)) THEN RETURN TRUE.
ELSE RETURN FALSE.
END CHECK_NS.

CHECK_NR PROCEDURE BYTE.
/* Check the Nr field of the received frame If Ns(P)+1=Nr(S) then the frame has been acknowledged else if Ns(P)=Nr(S) then the frame has not been acknowledged else reset the secondary */

IF (((RSD(STATION_NUMBER) NS = 1) AND O7H) = SHR(RCB,5)) THEN DO.
RSD(STATION_NUMBER) NS=((RSD(STATION_NUMBER) NS+1) AND O7H).
RSD(STATION_NUMBER) INFO_LENGTH=0.
END.

ELSE RETURN TRUE.

END CHECK_NR.

RECEIVE PROCEDURE
DECLARE 1 BYTE AUXILIARY.

RSD(STATION_NUMBER) BUFFER_STATUS=BUFFER_READY.
/* If an RNR was received buffer_status will be changed in the supervisory frame decode section further down in this procedure. any other response means the remote stations buffer is ready */

IF (RCS AND O1H)=0 THEN DO. /* I Frame Received */
IF (CHECK_NS=TRUE AND BDV=0 AND CHECK_NR=TRUE) THEN DO.
RSD(STATION_NUMBER) NR=((RSD(STATION_NUMBER) NR+1) AND O7H).
RSP=1.

19-73
PL/M-51 COMPILER
RUP1-44 Primary Station

100 : RECVD FIELD_LENGTH=RFL-1.

101 : IF RCB AND OPH=01H THEN DO. /* Supervisor frame received */

103 : IF CHECK_NR=FALSE THEN RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC.

107 : ELSE IF ((RCB AND OPH)=05H) THEN RNR /*

109 : THEN RSD(STATION_NUMBER) BUFFER_STATUS=BUFFER_NOT_READY.

110 : ELSE DO. /* Unnumbered frame or unknown frame received */

111 : IF RCB=FRMR THEN DO. /* If FRMR was received check for an
csacknowledged I frame */

113 : RCB=SIU_RECV_BUFFER(1); I=CHECK_NR;

115 : END.

116 : RSD(STATION_NUMBER) STATION_STATE=GO_TO_DISC.

117 : END.

118 : END RECEIVE.

120 : XMIT_I_T S PROCEDURE (TEMP).

121 : DECLARE TEMP BYTE.

122 : IF TEMP=I_FRAME THEN DO. /* Transmit I frame */

124 : DO TEMP=0 TO RSD(STATION_NUMBER).INFO_LENGTH-1.

126 : SIU_XMIT_BUFFER(TEMP)=RSD(STATION_NUMBER) DATA(TEMP).

127 : /* Build the I frame control field */

128 : TEMP=(SHL(RSD(STATION_NUMBER) NR.S) OR SHL(RSD(STATION_NUMBER) NS.1) OR 10H).

130 : TBL=RSD(STATION_NUMBER).INFO_LENGTH.

132 : CALL XMIT(TEMP).

133 : IF TIME_OUT=FALSE THEN CALL RECEIVE.

134 : ELSE DO. /* Transmit RR or RNR */

136 : ELSE TEMP=RNR.

19-74
TEMP = (SHL(RSD(STATION_NUMBER) NR, 5) OR TEMP).
TBL = 0:
CALL XMIT(TEMP).
IF TIME_OUT_FALSE
THEN CALL RECEIVE.
END.
BUFFER_TRANSFER PROCEDURE.
DECLARE I BYTE AUXILIARY, J BYTE AUXILIARY;
DO I = 0 TO NUMBER_OF_STATIONS-1.
IF RSD(I) STATION_ADDRESS = SIU_RECV_BUFFER(0)
THEN GOTO T1;
END.
T1: IF I = NUMBER_OF_STATIONS /* If the addressed station does not exist,
    then discard the data */
    THEN DO.
    RB = 0,
    RETURN;
END.
ELSE IF RSD(I) INFO_LENGTH = 0
THEN DO.
    RSD(I) INFO_LENGTH = RECV_FIELD_LENGTH.
    DO J = 1 TO RECV_FIELD_LENGTH.
    RSD(I) DATA(J-1) = SIU_RECV_BUFFER(J).
    END.
    RB = 0;
END.
END BUFFER_TRANSFER;
BEGIN.
CALL POWER_ON.
DO FOREVER.
DO STATION_NUMBER = 0 TO NUMBER_OF_STATIONS-1.
STAD = RSD(STATION_NUMBER) STATION_ADDRESS,
IF RSD(STATION_NUMBER) STATION_STATE = DISCONNECT_S
THEN CALL SEND_SNRM.
ELSE IF RSD(STATION_NUMBER) STATION_STATE = GO_TO_DISC
THEN CALL SEND_01.
ELSE IF RSD(STATION_NUMBER) INPO_LENGTH = 0
AND (RSD(STATION_NUMBER) BUFFER_STATUS = BUFFER_READY)
THEN CALL XMIT_I_T_S(T_FRAME).
ELSE IF RB = 0
THEN CALL XMIT_I_T_S(T_RRR).
ELSE CALL XMIT_I_T_S(T_BRR).
IF RB = 1
THEN CALL BUFFER_TRANSFER.
END.
END MAINMOD.

WARNINGS
1 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION

code size = 033H 1341D
constant size = 003H 2D
Direct variable size = 40H+02H 64H+ 2D
Indirect variable size = 40H+00H 64H+ 0D
Bit size = 01H+00H 1D+ 0D
Bit-addressable size = 00H+00H 5D+ 0D
Auxiliary variable size = 00F3H 147D
Maximum stack size = 00F9H 25D
Register-bank(s) used = 0 1
456 lines read
0 program error(s)
End of PL/M-51 compilation
8044AH/8344AH
HIGH PERFORMANCE 8-BIT MICROCONTROLLER
WITH ON-CHIP SERIAL COMMUNICATION CONTROLLER

- 8044AH — CPU/SIU with Factory Mask Programmable ROM
- 8344AH — An 8044AH used with External Program Memory
- 8744H — An 8044AH with User Programmable/Erasable EPROM

8051 MICROCONTROLLER CORE
- Optimized for Real Time Control
  12 MHz Clock, Priority Interrupts,
  32 Programmable I/O lines,
  Two 16-bit Timer/Counters
- Boolean Processor
- 4K x 8 ROM, 192 x 8 RAM
- 64K Accessible External Program Memory
- 64K Accessible External Data Memory
- 4 \( \mu \)s Multiply and Divide

The RUPI-44 family integrates a high performance 8-bit Microcontroller, the Intel 8051 Core, with an intelligent/high performance HDLC/SDLC serial communication controller, called the Serial Interface Unit (SIU). See Figure 1. This dual architecture allows complex control and high speed data communication functions to be realized cost effectively.

Specifically, the 8044's Microcontroller features: 4K byte On-Chip program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source, 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The Serial Interface Unit (SIU) manages the interface to a high speed serial link. The SIU offloads the On-Chip 8051 Microcontroller of communication tasks, thereby freeing the CPU to concentrate on real time control tasks.

The RUPI-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUPI-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOsII technology and packaged in a 40-pin DIP.

The 8344 is available in two versions: C8344H and P8344AH. Refer to the 8744 data sheet for the C8344H.

8044's Dual Controller Architecture

Figure 1. Dual Controller Architecture

January 1985
Table 1. RUPI™-44 Family Pin Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSS</td>
<td>Circuit ground potential.</td>
</tr>
<tr>
<td>VCC</td>
<td>+5V power supply during operation and program verification.</td>
</tr>
<tr>
<td>PORT 0</td>
<td>Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.</td>
</tr>
<tr>
<td>PORT 1</td>
<td>Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads. In non-loop mode two of the I/O lines serve alternate functions: - RTS (P1.6), Request-to-Send output. A low indicates that the RUPI-44 is ready to transmit. - CTS (P1.7), Clear-to-Send input. A low indicates that a receiving station is ready to receive.</td>
</tr>
<tr>
<td>PORT 2</td>
<td>Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.</td>
</tr>
<tr>
<td>PORT 3</td>
<td>Port 3 is an 8-bit bidirectional I/O port. It also contains the interrupt, timer, serial port and RD and WR pins that are used by various options. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads. In addition to I/O, some of the pins also serve alternate functions as follows: - RXD (P3.0), In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes. - TxD (P3.1), In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode. - INT0 (P3.2), Interrupt 0 input or gate control input for counter 0. - INT1 (P3.3), Interrupt 1 input or gate control input for counter 1. - TO (P3.4), Input to counter 0.</td>
</tr>
<tr>
<td>SCLK T1 (P3.5), In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.</td>
<td></td>
</tr>
<tr>
<td>WR (P3.6), The write control signal latches the data byte from Port 0 into the External Data Memory.</td>
<td></td>
</tr>
<tr>
<td>RD (P3.7), The read control signal enables External Data Memory to Port 0.</td>
<td></td>
</tr>
<tr>
<td>RST</td>
<td>A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (≈8.2kΩ) from RST to VSS permits power-on reset when a capacitor (≈10μF) is also connected from this pin to VCC.</td>
</tr>
<tr>
<td>ALE/PROG</td>
<td>Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.</td>
</tr>
<tr>
<td>PSEN</td>
<td>The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.</td>
</tr>
<tr>
<td>EA/NPP</td>
<td>When held at a TTL high level, the RUPI-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPI-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744.</td>
</tr>
<tr>
<td>XTAL 1</td>
<td>Input to the oscillator’s high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.</td>
</tr>
<tr>
<td>XTAL 2</td>
<td>Output from the oscillator’s amplifier. Input to the internal timing circuitry. A crystal or external source can be used.</td>
</tr>
</tbody>
</table>
Figure 2. Logic Symbol

Figure 3. Pin Configuration

Figure 4. Block Diagram
Functional Description

General

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Communication Controller to provide a single-chip solution which will efficiently implement a distributed processing or distributed control system. The microcontroller is a self-sufficient unit containing ROM, RAM, ALU, and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The 8044 replaces the 8051's serial interface with an intelligent SDLC/HDLC Serial Interface Unit (SIU). 64 more bytes of RAM have been added to the 8051 RAM array. The SIU can communicate at bit rates up to 2.4 M bps. The SIU works concurrently with the Microcontroller so that there is no throughput loss in either unit. Since the SIU possesses its own intelligence, the CPU is off-loaded from many of the communications tasks, thus dedicating more of its computing power to controlling local peripherals or some external process.

The Microcontroller

The microcontroller is a stand-alone high-performance single-chip computer intended for use in sophisticated real-time application such as instrumentation, industrial control, and intelligent computer peripherals.

The major features of the microcontroller are:
- 8-bit CPU
- on-chip oscillator
- 4K bytes of ROM
- 192 bytes of RAM
- 32 I/O lines
- 64K address space for external Data Memory
- 64K address space for external Program Memory
- two fully-programmable 16-bit timer/counters
- a five-source interrupt structure with two priority levels
- bit addressability for Boolean processing
- 1 μsec instruction cycle time for 60% of the instructions 2 μsec instruction cycle time for 40% of the instructions
- 4 μsec cycle time for 8 by 8 bit unsigned Multiply/Divide

Internal Data Memory

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 5.

Parallel I/O

The 8044 has 32 general-purpose I/O lines which are arranged into four groups of eight lines. Each group is called a port. Hence there are four ports; Port 0, Port 1, Port 2, and Port 3. Up to five lines from 1 and Port 2 are dedicated to supporting the serial channel when the SIU is invoked. Due to the nature of the serial port, two of Port 3's I/O lines (P3.0 and P3.1) do not have latched outputs. This is true whether or not the serial channel is used.
### ARITHMETIC OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A,Rn</td>
<td>Add register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, Rn</td>
<td>Add direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, @Ri</td>
<td>Add indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADD A, #data</td>
<td>Add immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A,Rn</td>
<td>Add register to Accumulator with Carry</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, direct</td>
<td>Add direct byte to A with Carry flag</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, @Ri</td>
<td>Add indirect RAM to A with Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDC A, #data</td>
<td>Add immediate data to A with Carry flag</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

### LOGICAL OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL A,Rn</td>
<td>AND register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, direct</td>
<td>AND direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, @Ri</td>
<td>AND indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL A, #data</td>
<td>AND immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ANL direct, A</td>
<td>AND Accumulator to direct byte</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ANL direct, #data</td>
<td>AND immediate data to direct byte</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>ORL A,Rn</td>
<td>OR register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, direct</td>
<td>OR direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

### LOGICAL OPERATIONS (CONTINUED)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Destination</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORL A, @Ri</td>
<td>OR indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ORL A, #data</td>
<td>OR immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ORL direct, A</td>
<td>OR Accumulator to direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ORL direct, #data</td>
<td>OR immediate data to direct byte</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>XRL A, Rn</td>
<td>Exclusive-OR register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, direct</td>
<td>Exclusive-OR direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, @Ri</td>
<td>Exclusive-OR indirect RAM to A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XRL A, #data</td>
<td>Exclusive-OR immediate data to A</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

### DATA TRANSFER

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,Rn</td>
<td>Move register to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, direct</td>
<td>Move direct byte to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, @Ri</td>
<td>Move indirect RAM to Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV A, #data</td>
<td>Move immediate data to Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV Rn, A</td>
<td>Move Accumulator to register</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV Rn, direct</td>
<td>Move direct byte to register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV Rn, #data</td>
<td>Move immediate data to register</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV direct, A</td>
<td>Move Accumulator to direct byte</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV direct, Rn</td>
<td>Move register to direct byte</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV direct, direct</td>
<td>Move direct byte to direct</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>MOV direct, @Ri</td>
<td>Move indirect RAM to direct byte</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
### DATA TRANSFER (CONTINUED)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV direct,#data</td>
<td>Move immediate data to direct byte</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>MOV @Ri,A</td>
<td>Move Accumulator to indirect RAM</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MOV @Ri,direct</td>
<td>Move direct byte to indirect RAM</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV @Ri,#data</td>
<td>Move immediate data to indirect RAM</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>MOV DPTR,#data16</td>
<td>Load Data Pointer with a 16-bit constant</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>MOVC A,@A+DPTR</td>
<td>Move Code byte relative to DPTR to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVC A,@A+PC</td>
<td>Move Code byte relative to PC to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVC A,@Ri</td>
<td>Move External RAM (8-bit addr) to A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVC A,@DPTR</td>
<td>Move External RAM (16-bit addr) to A</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOVC A,PC</td>
<td>Move A to External RAM (8-bit addr)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOVC A,APTR,A</td>
<td>Move A to External RAM (16-bit addr)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>PUSH direct</td>
<td>Push direct byte onto stack</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>POP direct</td>
<td>Pop direct byte from stack</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>XCH A,Rn</td>
<td>Exchange register with Accumulator</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XCH A,direct</td>
<td>Exchange direct byte with Accumulator</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>XCH A,@Ri</td>
<td>Exchange indirect RAM with A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>XCHD A,@Ri</td>
<td>Exchange low-order Digit indirect RAM w A</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### BOOLEAN VARIABLE MANIPULATION

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR C</td>
<td>Clear Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CLR bit</td>
<td>Clear direct bit</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SETB bit</td>
<td>Set Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SETB bit</td>
<td>Set direct Bit</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>CPL C</td>
<td>Complement Carry flag</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CPL bit</td>
<td>Complement direct bit</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ANL C,bit</td>
<td>AND direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ANL C,bit</td>
<td>AND complement of direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL C,bit</td>
<td>OR direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ORL C,bit</td>
<td>OR complement of direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV C,bit</td>
<td>Move direct bit to Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MOV bit,C</td>
<td>Move Carry flag to direct bit</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### PROGRAM AND MACHINE CONTROL

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Cyc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACALL addr11</td>
<td>Absolute Subroutine Call</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>LCALL addr16</td>
<td>Long Subroutine Call</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>RET</td>
<td>Return from subroutine</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RETI</td>
<td>Return from interrupt</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>AJMP addr11</td>
<td>Absolute Jump</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>LJMP addr16</td>
<td>Long Jump</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>SJMP rel</td>
<td>Short Jump (relative addr)</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JMP @A+DPTR</td>
<td>Jump indirect relative to the DPTR</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>JZ rel</td>
<td>Jump if accumulator is Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNZ rel</td>
<td>Jump if accumulator is Not Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JC rel</td>
<td>Jump if Carry flag is set</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JNC rel</td>
<td>Jump if No Carry flag</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>JB bit,rel</td>
<td>Jump if direct Bit set</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>JNB bit,rel</td>
<td>Jump if direct Bit Not set</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>JBC bit,rel</td>
<td>Jump if direct Bit is set &amp; Clear bit</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CJNE A,direct,rel</td>
<td>Compare direct to A &amp; Jump if Not Equal</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CJNE A,#data,rel</td>
<td>Comp, immed, to A &amp; Jump if Not Equal</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CJNE Rn,#data,rel</td>
<td>Comp, immed, to reg &amp; Jump if Not Equal</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ Rn,rel</td>
<td>Decrement register &amp; Jump if Not Zero</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DJNZ direct,rel</td>
<td>Decrement direct &amp; Jump if Not Zero</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**Notes on data addressing modes:**
- **Rn** — Working register R0-R7
- **direct** — 128 internal RAM locations, any I/O port, control or status register
- **@Ri** — Indirect internal RAM location addressed by register R0 or R1
- **#data** — 8-bit constant included in instruction
- **#data16** — 16-bit constant included as bytes 2 & 3 of instruction
- **bit** — 128 software flags, any I/O pin, control or status bit

**Notes on program addressing modes:**
- **addr16** — Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
- **Addr11** — Destination address for ACALL & AJMP will be within the same 2-K program memory as the first byte of the following instruction
- **rel** — SJMP and all conditional jumps include an 8-bit offset byte, range is +127–128 bytes relative to first byte of the following instruction

All mnemonics copyrighted © Intel Corporation 1979
Port 0 and Port 2 also have an alternate dedicated function. When placed in the external access mode, Port 0 and Port 2 become the means by which the 8044 communicates with external program memory. Port 0 and Port 2 are also the means by which the 8044 communicates with external data memory. Peripherals can be memory mapped into the address space and controlled by the 8044.

Timer/Counters

The 8044 contains two 16-bit counters which can be used for measuring time intervals, measuring pulse widths, counting events, generating precise periodic interrupt requests, and clocking the serial communications. Internally the Timers are clocked at 1/12 of the crystal frequency, which is the instruction cycle time. Externally the counters can run up to 500 KHz.

Interrupt System

External events and the real-time driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two priority level, nested interrupt system is provided. Interrupt response latency ranges from 3 μsec to 7 μsec when using a 12 MHz clock.

All five interrupt sources can be mapped into one of the two priority levels. Each interrupt source can be enabled or disabled individually or the entire interrupt system can be enabled or disabled. The five interrupt sources are: Serial Interface Unit, Timer 1, Timer 2, and two external interrupts. The external interrupts can be either level or edge triggered.

Serial Interface Unit (SIU)

The Serial Interface Unit is used for HDLC/SDLC communications. It handles Zero Bit Insertion/Deletion, Flags, automatic address recognition, and a 16-bit cyclic redundancy check. In addition it implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. In certain applications it is advantageous to have the CPU control the reception or transmission of every single frame. For this reason the SIU has two modes of operation: "AUTO" and "FLEXIBLE" (or "NON-AUTO"). It is in the AUTO mode that the SIU responds to SDLC frames without CPU intervention; whereas, in the FLEXIBLE mode the reception or transmission of every single frame will be under CPU control.

There are three control registers and eight parameter registers that are used to operate the serial interface. These registers are shown in Figure 5 and Figure 6. The control registers set the modes of operation and provide status information. The eight parameter registers buffer the station address, receive and transmit control bytes, and point to the on-chip transmit and receive buffers.

Data to be received or transmitted by the SIU must be buffered anywhere within the 192 bytes of on-chip RAM. Transmit and receive buffers are not allowed to "wrap around" in RAM; a "buffer end" is generated after address 191 is reached.

AUTO Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. All AUTO mode responses to the primary station will conform to IBM's SDLC definition. The advantages of the AUTO mode are that less software is required to implement a secondary station, and the hardware generated response to polls is much faster than doing it in software. However, the Auto mode can not be used at a primary station.

To transmit in the AUTO mode the CPU must load the Transmit Information Buffer, Transmit Buffer Start register, Transmit Buffer Length register, and set the Transmit Buffer Full bit. The SIU automatically responds to a poll by transmitting an information frame with the P/F bit in the control field set. When the SIU receives a positive acknowledgement from the primary station, it automatically increments the Ns field in the NSNR register and interrupts the CPU. A negative acknowledgement would cause the SIU to retransmit the frames.

To receive in the AUTO mode, the CPU loads the Receive Buffer Start register, the Receive Buffer Length register, clears the Receive Buffer Protect bit, and sets the Receive Buffer Empty bit. If the SIU is polled in this state, and the TBF bit indicates that the Transmit Buffer is empty, an automatic RR response will be generated. When a valid information frame is received the SIU will automatically increment Nr in the NSNR register and interrupt the CPU.

While in the AUTO mode the SIU can recognize and respond to the following commands without CPU intervention: 1 (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and UP (Unnumbered Poll). The SIU can generate
**REGISTER NAMES**

<table>
<thead>
<tr>
<th>SYMBOLIC ADDRESS</th>
<th>BIT ADDRESS</th>
<th>BYTE ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B REGISTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACCUMULATOR</td>
<td>231 through 224</td>
<td>222 (D0H)</td>
</tr>
<tr>
<td>*THREE BYTE FIFO</td>
<td>223 through 222</td>
<td>223 (DFH)</td>
</tr>
<tr>
<td>FIFO</td>
<td>221 through 221</td>
<td>221 (DFH)</td>
</tr>
<tr>
<td>FIFO</td>
<td>220 through 220</td>
<td>220 (DFH)</td>
</tr>
<tr>
<td>TRANSMIT BUFFER START</td>
<td>219 through 219</td>
<td>219 (DBH)</td>
</tr>
<tr>
<td>TRANSMIT BUFFER LENGTH</td>
<td>218 through 218</td>
<td>218 (D8H)</td>
</tr>
<tr>
<td>TRANSMIT CONTROL BYTE</td>
<td>217 through 217</td>
<td>217 (D7H)</td>
</tr>
<tr>
<td>*SIU STATE COUNTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEND COUNT RECEIVE COUNT</td>
<td>216 through 216</td>
<td>216 (D6H)</td>
</tr>
<tr>
<td>PROGRAM STATUS WORD</td>
<td>215 through 215</td>
<td>215 (D5H)</td>
</tr>
<tr>
<td>*DMA COUNT</td>
<td>214 through 214</td>
<td>214 (D4H)</td>
</tr>
<tr>
<td>STATION ADDRESS</td>
<td>213 through 213</td>
<td>213 (D3H)</td>
</tr>
<tr>
<td>RECEIVE BUFFER LENGTH</td>
<td>212 through 212</td>
<td>212 (D2H)</td>
</tr>
<tr>
<td>RECEIVE BUFFER START</td>
<td>211 through 211</td>
<td>211 (D1H)</td>
</tr>
<tr>
<td>RECEIVE BUFFER LENGTH</td>
<td>210 through 210</td>
<td>210 (D0H)</td>
</tr>
<tr>
<td>RECEIVE CONTROL BYTE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SERIAL MODE</td>
<td>209 through 209</td>
<td>209 (C0H)</td>
</tr>
<tr>
<td>STATUS REGISTER</td>
<td>208 through 208</td>
<td>208 (C0H)</td>
</tr>
<tr>
<td>INTERRUPT ENABLE CONTROL</td>
<td>207 through 207</td>
<td>207 (C0H)</td>
</tr>
<tr>
<td>PORT 3</td>
<td>206 through 206</td>
<td>206 (C0H)</td>
</tr>
<tr>
<td>INTERRUPT ENABLE CONTROL</td>
<td>205 through 205</td>
<td>205 (C0H)</td>
</tr>
<tr>
<td>PORT 2</td>
<td>204 through 204</td>
<td>204 (C0H)</td>
</tr>
<tr>
<td>PORT 1</td>
<td>203 through 203</td>
<td>203 (C0H)</td>
</tr>
<tr>
<td>TIMER HIGH 1</td>
<td>202 through 202</td>
<td>202 (C0H)</td>
</tr>
<tr>
<td>TIMER HIGH 0</td>
<td>201 through 201</td>
<td>201 (C0H)</td>
</tr>
<tr>
<td>TIMER LOW 1</td>
<td>200 through 200</td>
<td>200 (C0H)</td>
</tr>
<tr>
<td>TIMER LOW 0</td>
<td>199 through 199</td>
<td>199 (C0H)</td>
</tr>
<tr>
<td>TIMER MODE</td>
<td>198 through 198</td>
<td>198 (C0H)</td>
</tr>
<tr>
<td>TIMER CONTROL</td>
<td>197 through 197</td>
<td>197 (C0H)</td>
</tr>
<tr>
<td>DATA POINTER HIGH</td>
<td>196 through 196</td>
<td>196 (C0H)</td>
</tr>
<tr>
<td>DATA POINTER LOW</td>
<td>195 through 195</td>
<td>195 (C0H)</td>
</tr>
<tr>
<td>STACK POINTER</td>
<td>194 through 194</td>
<td>194 (C0H)</td>
</tr>
<tr>
<td>PORT 0</td>
<td>193 through 193</td>
<td>193 (C0H)</td>
</tr>
</tbody>
</table>

*SFR's CONTAINING DIRECT ADDRESSABLE BITS

*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

**Figure 5. Mapping of Special Function registers**

**Figure 6. Serial Interface Unit Control registers**
the following responses without CPU intervention: I (Information), RR (Receive Ready), and RNR (Receive Not Ready).

When the Receive Buffer Empty bit (RBE) indicates that the Receive Buffer is empty, the receiver is enabled, and when the RBE bit indicates that the Receive Buffer is full, the receiver is disabled. Assuming that the Receive Buffer is empty, the SIU will respond to a poll with an I frame if the Transmit Buffer is full. If the Transmit Buffer is empty, the SIU will respond to a poll with a RR command if the Receive Buffer Protect bit (RBP) is cleared, or an RNR command if RBP is set.

**FLEXIBLE (or NON-AUTO) Mode**

In the FLEXIBLE mode all communications are under control of the CPU. It is the CPU’s task to encode and decode control fields, manage acknowledgements, and adhere to the requirements of the HDLC/SDLC protocols. The 8044 can be used as a primary or a secondary station in this mode.

To receive a frame in the FLEXIBLE mode, the CPU must load the Receive Buffer Start register, the Receive Buffer Length register, clear the Receive Buffer Protect bit, and set the Receive Buffer Empty bit. If a valid opening flag is received and the address field matches the byte in the Station Address register or the address field contains a broadcast address, the 8044 loads the control field in the receive control byte register, and loads the I field in the receive buffer. If there is no CRC error, the SIU interrupts the CPU, indicating a frame has just been received. If there is a CRC error, no interrupt occurs. The Receive Field Length register provides the number of bytes that were received in the information field.

To transmit a frame, the CPU must load the transmit information buffer, the Transmit Buffer Start register, the Transmit Buffer Length register, the Transmit Control Byte, and set the TBF and the RFS bit. The SIU, unsolicited by an HDLC/SDLC frame, will transmit the entire information frame, and interrupt the CPU, indicating the completion of transmission. For supervisory frames or unnumbered frames, the transmit buffer length would be 0.

**CRC**

The FCS register is initially set to all 1’s prior to calculating the FCS field. The SIU will not interrupt the CPU if a CRC error occurs (in both AUTO and FLEXIBLE modes). The CRC error is cleared upon receiving of an opening flag.

**Frame Format Options**

In addition to the standard SDLC frame format, the 8044 will support the frames displayed in Figure 7. The standard SDLC frame is shown at the top of this figure. For the remaining frames the information field will incorporate the control or address bytes and the frame check sequences; therefore these fields will be stored in the Transmit and Receive buffers. For example, in the non-buffered mode the third byte is treated as the beginning of the information field. In the non-addressed mode, the information field begins after the opening flag. The mode bits to set the frame format options are found in the Serial Mode register and the Status register.

**EXTENDED ADDRESSING**

To realize an extended control field or an extended address field using the HDLC protocol, the FLEXIBLE mode must be used. For an extended control field, the SIU is programmed to be in the non-buffered mode. The extended control field will be the first and second bytes in the Receive and Transmit Buffers. For extended addressing the SIU is placed in the non-addressed mode. In this mode the CPU must implement the address recognition for received frames. The addressing field will be the initial bytes in the Transmit and Receive buffers followed by the control field.

The SIU can transmit and receive only frames which are multiples of 8 bits. For frames received with other than 8-bit multiples, a CRC error will cause the SIU to reject the frame.

**SDLC Loop Networks**

The SIU can be used in an ADLC loop as a secondary or primary station. When the SIU is placed in the Loop mode it receives the data on pin 10 and transmits the data one bit time delayed on pin 11. It can also recognize the Go ahead signal and change it into a flag when it is ready to transmit. As a secondary station the SIU can be used in the AUTO or FLEXIBLE modes. As a primary station the FLEXIBLE mode is used; however, additional hardware is required for generating the Go Ahead bit pattern. In the Loop mode the maximum data rate is 1 Mbps clocked or 375 Kbps self-clocked.

**SDLC Multidrop Networks**

The SIU can be used in a SDLC non-loop configuration as a secondary or primary station. When the SIU is placed in the non-loop mode, data is received and transmitted on pin 11, and pin 10 drives a tri-state buffer. In non-loop mode, modem interface pins, RTS and CTS, become available.
### Data Clocking Options

The 8044's serial port can operate in an externally clocked or self clocked system. A clocked system provides to the 8044 a clock synchronization to the data. A self-clocked system uses the 8044's on-chip Digital Phase Locked Loop (DPLL) to recover the clock from the data, and clock this data into the Serial Receive Shift Register.

In this mode, a clock synchronized with the data is externally fed into the 8044. This clock may be generated from an External Phase Locked Loop, or possibly supplied along with the data. The 8044 can transmit and receive data in this mode at rates up to 2.4 Mbps.

This self clocked mode allows data transfer without a common system data clock. An on-chip Digital Phase Locked Loop is employed to recover the data clock which is encoded in the data stream. The DPLL will converge to the nominal bit center within eight bit transitions, worst case. The DPLL requires a reference clock of either 16 times (16x) or 32 times (32x) the data rate. This reference clock may be externally applied or internally generated. When internally generated either the 8044's internal logic clock (crystal frequency divided by two) or the timer 1 overflow is used as the reference clock. Using the internal timer 1 clock the data rates can vary from 244 to 62.5 Kbps. Using the internal logic clock at a 16x sampling rate, receive data rates can vary from 187.5 Kbps, or 375 Kbps. When the reference clock for the DPLL is externally applied the data rates can vary from 0 to 375 Kbps at a 16x sampling rate.

To aid in a Phase Locked Loop capture, the SIU has a NRZI (Non Return to Zero Inverted) data encoding and decoding option. Additionally the SIU has a pre-frame sync option that transmits two bytes of alternating 1's and 0's to ensure that the receive station DPLL will be synchronized with the data by the time it receives the opening flag.
**Control and Status Registers**

There are three SIU Control and Status Registers:

- **Serial Mode Register (SMD)**
- **Status/Command Register (STS)**
- **Send/Receive Count Register (NSNR)**

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below.

### SMD: Serial Mode Register (byte-addressable)

**Bit: 7 6 5 4 3 2 1 0**

<table>
<thead>
<tr>
<th>SCM2</th>
<th>SCM1</th>
<th>SCM0</th>
<th>NRZI</th>
<th>LOOP</th>
<th>PFS</th>
<th>NB</th>
<th>NFCS</th>
</tr>
</thead>
</table>

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMD.0</td>
<td>NFCS</td>
<td>No FCS field in the SDLC frame.</td>
</tr>
<tr>
<td>SMD.1</td>
<td>NB</td>
<td>Non-Buffered mode. No control field in the SDLC frame.</td>
</tr>
<tr>
<td>SMD.2</td>
<td>PFS</td>
<td>Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.</td>
</tr>
<tr>
<td>SMD.3</td>
<td>LOOP</td>
<td>Loop configuration.</td>
</tr>
<tr>
<td>SMD.4</td>
<td>NRZI</td>
<td>NRZI coding option.</td>
</tr>
<tr>
<td>SMD.5</td>
<td>SCM0</td>
<td>Select Clock Mode — Bit 0</td>
</tr>
<tr>
<td>SMD.6</td>
<td>SCM1</td>
<td>Select Clock Mode — Bit 1</td>
</tr>
<tr>
<td>SMD.7</td>
<td>SCM2</td>
<td>Select Clock Mode — Bit 2</td>
</tr>
</tbody>
</table>

The SCM bits decode as follows:

<table>
<thead>
<tr>
<th>SCM</th>
<th>Clock Mode</th>
<th>Data Rate (Bits/sec)*</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>Externally clocked</td>
<td>0-2.4M**</td>
</tr>
<tr>
<td>0 0 1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>Self clocked, timer overflow</td>
<td>244-62.5K</td>
</tr>
<tr>
<td>0 1 1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>Self clocked, external 16x</td>
<td>0-375K</td>
</tr>
<tr>
<td>1 0 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bit: 7 6 5 4 3 2 1 0**

<table>
<thead>
<tr>
<th>TBF</th>
<th>RBE</th>
<th>RTS</th>
<th>SI</th>
<th>BOV</th>
<th>OPB</th>
<th>AM</th>
<th>RBP</th>
</tr>
</thead>
</table>

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B, C') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

<table>
<thead>
<tr>
<th>Bit #</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STS.0</td>
<td>RBP</td>
<td>Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.</td>
</tr>
<tr>
<td>STS.1</td>
<td>AM</td>
<td>AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.</td>
</tr>
<tr>
<td>STS.2</td>
<td>OPB</td>
<td>Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.</td>
</tr>
<tr>
<td>STS.3</td>
<td>BOV</td>
<td>Receive Buffer Overrun. BOV may be set or cleared by the SIU.</td>
</tr>
<tr>
<td>STS.4</td>
<td>SI</td>
<td>SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.</td>
</tr>
<tr>
<td>STS.5</td>
<td>RTS</td>
<td>Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.</td>
</tr>
</tbody>
</table>

**8044AH/8344AH ADVANCE INFORMATION**

20-11
NSNR: Send/Receive Count Register (byte-addressable)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>NSNR.7</td>
<td>Receive Sequence Counter—Bit 7</td>
</tr>
<tr>
<td>6</td>
<td>NSNR.6</td>
<td>Receive Sequence Counter—Bit 6</td>
</tr>
<tr>
<td>5</td>
<td>NSNR.5</td>
<td>Receive Sequence Counter—Bit 5</td>
</tr>
<tr>
<td>4</td>
<td>NSNR.4</td>
<td>Receive Sequence Counter—Bit 4</td>
</tr>
<tr>
<td>3</td>
<td>NSNR.3</td>
<td>Receive Sequence Counter—Bit 3</td>
</tr>
<tr>
<td>2</td>
<td>NSNR.2</td>
<td>Receive Sequence Counter—Bit 2</td>
</tr>
<tr>
<td>1</td>
<td>NSNR.1</td>
<td>Receive Sequence Counter—Bit 1</td>
</tr>
<tr>
<td>0</td>
<td>NSNR.0</td>
<td>Receive Sequence Counter—Bit 0</td>
</tr>
</tbody>
</table>

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles (‘JBC /B, REL’, and ‘MOV /B,C’) should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS = 0 and RBE = 0). Normally, STAD is accessed only during initialization.

TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF = 0).

TBL: Transmit Buffer Length Register (byte-addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL = 0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF = 0).

NOTE: The transmit and receive buffers are not allowed to “wrap around” in the on-chip RAM. A “buffer end” is automatically generated if address 191 (BFH) is reached.

TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF = 0). The NS and NR counters are not used in the NON-AUTO mode.

RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE = 0).

RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip RAM allocated for the received I-field. RBL = 0 is valid. The CPU should write RBL only when RBE = 0.

RFL: Receive Field Length Register (byte-addressable)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL = 0 is valid. RFL should be accessed by the CPU only when RBE = 0.
RCB: Receive Control Byte Register
(byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE = 0.

ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can exercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register. This register provides a useful means for debugging 8044 receiver problem.
**ABSOLUTE Maximum Ratings**

Ambient Temperature Under Bias: 0 to 70°C
Storage Temperature: -65°C to +150°C
Voltage on Any Pin With Respect to Ground (Vss): -0.5V to +7V
Power Dissipation: 2 Watts

*Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

**DC Characteristics** \((T_A = 0°C \text{ to } 70°C, VCC = 5V \pm 10%, VSS = 0V)\)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Typ</th>
<th>Max</th>
<th>Units</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (Except RST/VPD and XTAL2)</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage To RST/VPD For Reset, XTAL2</td>
<td>2.5</td>
<td>V</td>
<td>XTAL1 to VSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage Ports 1, 2, 3 (Note 1)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 1.6mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage Port 0 ALE, (\overline{PSEN}) (Note 1)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 3.2mA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage Ports 1, 2, 3</td>
<td>2.4</td>
<td>V</td>
<td>IOH = -80μA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage Port 0, ALE, (\overline{PSEN})</td>
<td>2.4</td>
<td>V</td>
<td>IOH = -400μA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Logical 0 Input Current Ports 1, 2, 3</td>
<td>-800</td>
<td>μA</td>
<td>XTAL1 at VSS VIL = 0.45V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IILH1</td>
<td>Input High Current (T0) RST/VPD For Reset</td>
<td>500</td>
<td>μA</td>
<td>Vin = VCC - 1.5V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIL2</td>
<td>Input Leakage Current To Port 0, (\overline{EA})</td>
<td>10</td>
<td>μA</td>
<td>0.45V &lt; Vin &lt; VCC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current</td>
<td>125</td>
<td>200</td>
<td>mA</td>
<td>TA = 25°C</td>
<td></td>
</tr>
<tr>
<td>CIO</td>
<td>Capacitance of I/O Buffer</td>
<td>10</td>
<td>pF</td>
<td>fc = 1MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IIL2</td>
<td>Logical 0 Input Current XTAL 2</td>
<td>-3.5</td>
<td>mA</td>
<td>XTAL1 = VSS VIL = 0.45V</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note 1:** VOL is degraded when the RUPI-44 rapidly discharges external capacitance. This A.C. noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the RUPI-44 as possible.
A.C. CHARACTERISTICS (TA 0°C to 70°C, VCC = 5V ± 10%; τSS = OV, CL for Port 0, ALE and PSEN Outputs = 100pF; CL for All Other Outputs = 80 pF)

**Program Memory**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Clock</th>
<th>Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>TLHLL</td>
<td>ALE Pulse Width</td>
<td>127</td>
<td>ns</td>
</tr>
<tr>
<td>TAVLL</td>
<td>Address Setup to ALE</td>
<td>43</td>
<td>ns</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold After ALE</td>
<td>48</td>
<td>ns</td>
</tr>
<tr>
<td>TLLIV</td>
<td>ALE To Valid Instr In</td>
<td>233</td>
<td>ns</td>
</tr>
<tr>
<td>TLLPL</td>
<td>ALE To PSEN</td>
<td>58</td>
<td>ns</td>
</tr>
<tr>
<td>TPLPH</td>
<td>PSEN Pulse Width</td>
<td>215</td>
<td>ns</td>
</tr>
<tr>
<td>TPLIV</td>
<td>PSEN To Valid Instr In</td>
<td>125</td>
<td>ns</td>
</tr>
<tr>
<td>TPXIX</td>
<td>Input Instr After PSEN</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TPXIZ</td>
<td>Input Instr Float After PSEN</td>
<td>63</td>
<td>ns</td>
</tr>
<tr>
<td>TPXAZ</td>
<td>Address Valid After PSEN</td>
<td>75</td>
<td>ns</td>
</tr>
<tr>
<td>TAVIV</td>
<td>Address To Valid Instr In</td>
<td>302</td>
<td>ns</td>
</tr>
<tr>
<td>TAZPL</td>
<td>Address Float To PSEN</td>
<td>25</td>
<td>ns</td>
</tr>
</tbody>
</table>

**Notes:**
1. TLLAX for access to program memory is different from TLLAX for data memory.
2. Interfacing RUPI-44 devices with float times up to 75ns is permissible. This limited bus contention will not cause any damage to Port 0 drivers.

**External Data Memory**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Clock</th>
<th>Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>TRRLH</td>
<td>RD Pulse Width</td>
<td>400</td>
<td>ns</td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR Pulse Width</td>
<td>400</td>
<td>ns</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold After ALE</td>
<td>48</td>
<td>ns</td>
</tr>
<tr>
<td>TRLDV</td>
<td>RD To Valid Data In</td>
<td>250</td>
<td>ns</td>
</tr>
<tr>
<td>TRHDX</td>
<td>Data Hold After RD</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>TRHZD</td>
<td>Data Float After RD</td>
<td>97</td>
<td>ns</td>
</tr>
<tr>
<td>TLLDV</td>
<td>ALE To Valid Data In</td>
<td>517</td>
<td>ns</td>
</tr>
<tr>
<td>TAVDV</td>
<td>Address To Valid Data In</td>
<td>585</td>
<td>ns</td>
</tr>
<tr>
<td>TLLWL</td>
<td>ALE To WR or RD</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td>TAVWL</td>
<td>Address To WR or RD</td>
<td>203</td>
<td>ns</td>
</tr>
<tr>
<td>TWHLH</td>
<td>WR or RD High To ALE High</td>
<td>43</td>
<td>ns</td>
</tr>
<tr>
<td>TVWXX</td>
<td>Data Valid To WR Transition</td>
<td>33</td>
<td>ns</td>
</tr>
<tr>
<td>TQWWH</td>
<td>Data Setup Before WR</td>
<td>433</td>
<td>ns</td>
</tr>
<tr>
<td>TWHQX</td>
<td>Data Hold After WR</td>
<td>33</td>
<td>ns</td>
</tr>
<tr>
<td>TRLAZ</td>
<td>Address Float After RD</td>
<td>0</td>
<td>ns</td>
</tr>
</tbody>
</table>

**Notes:**
1. TLLAX for access to program memory is different from TLLAX for access data memory

**Serial Interface**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDCY</td>
<td>Data Clock</td>
<td>420</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TDCL</td>
<td>Data Clock Low</td>
<td>180</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TDCH</td>
<td>Data Clock High</td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>
Serial Interface (Continued)

<table>
<thead>
<tr>
<th>ITD</th>
<th>Transmit Data Delay</th>
<th>180</th>
<th>ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDSS</td>
<td>Data Setup Time</td>
<td>40</td>
<td>ns</td>
</tr>
<tr>
<td>TDHS</td>
<td>Data Hold Time</td>
<td>40</td>
<td>ns</td>
</tr>
</tbody>
</table>

WAVEFORMS

Memory Access

Program Memory Read Cycle

Data Memory Read Cycle

Data Memory Write Cycle

20-16
SERIAL I/O WAVEFORMS

Synchronous Data Transmission

- **SCLK**
- **DATA**

Synchronous Data Reception

- **SCLK**
- **DATA**
AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS

AC testing inputs are driven at 2.4V for a logic “1” and 0.45V for a logic “0”.
Timing measurements are made at 2.0V for a logic “1” and 0.8V for a logic “0”.

EXTERNAL CLOCK DRIVE XTAL2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Variable Clock Frequency (1.2 MHz to 12 MHz)</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLCL</td>
<td>Oscillator Period</td>
<td>Min 83.3  Max 287.5 ns</td>
<td></td>
</tr>
<tr>
<td>TCHCX</td>
<td>High Time</td>
<td>Min 20 ns</td>
<td></td>
</tr>
<tr>
<td>TCLCX</td>
<td>Low Time</td>
<td>Min 20 ns</td>
<td></td>
</tr>
<tr>
<td>TCLCH</td>
<td>Rise Time</td>
<td>Min 20 ns</td>
<td></td>
</tr>
<tr>
<td>TCHCL</td>
<td>Fall Time</td>
<td>Min 20 ns</td>
<td></td>
</tr>
</tbody>
</table>
This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (TA = 25°C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.
8744H
HIGH PERFORMANCE 8-BIT MICROCONTROLLER
WITH On-CHIP SERIAL COMMUNICATION CONTROLLER

8051 MICROCONTROLLER CORE
- Optimized for Real Time Control
  12 MHz Clock, Priority Interrupts,
  32 Programmable I/O lines,
  Two 16-bit Timer/Counters
- Boolean Processor
- 4K x 8 EPROM, 192 x 8 RAM
- 64K Accessible External Program Memory
- 64K Accessible External Data Memory
- 4 µs Multiply and Divide

SERIAL INTERFACE UNIT (SIU)
- Serial Communication Processor that
  Operates Concurrently to CPU
- 2.4 Mbps Maximum Data Rate
- 375 Kbps using On-Chip Phase
  Locked Loop
- Communication Software in Silicon:
  — Complete Data Link Functions
  — Automatic Station Responses
- Operates as an SDLC Primary or Secondary Station

The RUPI-44 family integrates a high performance 8-bit Microcontroller, the Intel 8051 Core, with an intelligent/high performance HDLC/SDLC serial communication controller, called the Serial Interface Unit (SIU). See Figure 1. This dual architecture allows complex control and high speed data communication functions to be realized cost effectively.

Specifically, the 8044's Microcontroller features: 4K byte On-Chip program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The Serial Interface Unit (SIU) manages the interface to a high speed serial link. The SIU offloads the On-Chip 8051 Microcontroller of communication tasks, thereby freeing the CPU to concentrate on real time control tasks.

The RUPI-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUPI-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOSII technology and packaged in a 40-pin DIP.

The 8744H is available in a hermetically sealed, ceramic, 40-lead dual in-line package which includes a window that allows for EPROM erasure when exposed to ultraviolet light (See Erasure Characteristics). During normal operation, ambient light may adversely affect the functionality of the chip. Therefore, applications which expose the 8744H to ambient light may require an opaque label over the window.

8044’s Dual Controller Architecture

![Figure 1. Dual Controller Architecture](image)

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied

* INTEL CORPORATION, 1982

ORDER NUMBER: 210735-002
Table 1. RUPI™·44 Family Pin Description

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSS</td>
<td>Circuit ground potential.</td>
</tr>
<tr>
<td>VCC</td>
<td>+5V power supply during operation and program verification.</td>
</tr>
<tr>
<td>PORT 0</td>
<td>Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.</td>
</tr>
</tbody>
</table>
| PORT 1| Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads. In non-loop mode two of the I/O lines serve alternate functions:  
|       | - RTS (P1.6). Request-to-Send output. A low indicates that the RUPI-44 is ready to transmit. |
|       | - CTS (P1.7) Clear-to-Send input. A low indicates that a receiving station is ready to receive. |
| PORT 2| Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads. |
| PORT 3| Port 3 is an 8-bit quasi-bidirectional I/O port. It also contains the interrupt, timer, serial port and RD and WR pins that are used by various options. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads. In addition to I/O, some of the pins also serve alternate functions as follows:  
|       | - I/O RxD (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.  
|       | - DATA TxD (P3.1) In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode.  
|       | - INTO (P3.2). Interrupt 0 input or gate control input for counter 0.                           |
|       | - INI (P3.3). Interrupt 1 input or gate control input for counter 1.                          |
|       | - TO (P3.4). Input to counter 0.                                                               |
| RST   | A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (~8.2kΩ) from RST to VSS permits power-on reset when a capacitor (~10μf) is also connected from this pin to VCC. |
| ALE/PROG | Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version. |
| PSEN  | The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution. |
| EAVP  | When held at a TTL high level, the RUPI-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPI-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744. |
| XTAL 1| Input to the oscillator’s high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2. |
| XTAL 2| Output from the oscillator’s amplifier. Input to the internal timing circuitry. A crystal or external source can be used. |
Figure 2. Logic Symbol

Figure 3. Pin Configuration

Figure 4. Block Diagram
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ........... 0°C to 70°C
Storage Temperature ...................... -65°C to +150°C
Voltage On Any Pin to VSS (Except Vpp) .......................... -0.5V to +7V
Voltage from Vpp to VSS .......................... 21.5V
Power Dissipation .......................... 2 W

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS: (TA = 0°C to 70°C; VCC = 4.5V to 5.5V, VSS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
<th>Test Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIL</td>
<td>Input Low Voltage (except EA)</td>
<td>-0.5</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIL1</td>
<td>Input Low Voltage to EA</td>
<td>0</td>
<td>0.8</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH</td>
<td>Input High Voltage (Except XTAL2, RST)</td>
<td>2.0</td>
<td>VCC + 0.5</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>VIH1</td>
<td>Input High Voltage to XTAL2, RST</td>
<td>2.5</td>
<td>VCC + 0.5</td>
<td>V</td>
<td>XTAL1 = VSS</td>
</tr>
<tr>
<td>VOL</td>
<td>Output Low Voltage Ports 1, 2, 3 (Note 1)</td>
<td>0.45</td>
<td>V</td>
<td>IOL = 1.6mA</td>
<td></td>
</tr>
<tr>
<td>VOL1</td>
<td>Output Low Voltage Port 0, ALE, PSEN (Note 1)</td>
<td>0.60</td>
<td>V</td>
<td>IOL = 3.2mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.45</td>
<td>V</td>
<td>IOL = 2.4mA</td>
<td></td>
</tr>
<tr>
<td>VOH</td>
<td>Output High Voltage Ports 1, 2, 3</td>
<td>2.4</td>
<td>V</td>
<td>IOH = 80μA</td>
<td></td>
</tr>
<tr>
<td>VOH1</td>
<td>Output High Voltage Port 0 (in External Bus Mode), ALE, PSEN</td>
<td>2.4</td>
<td>V</td>
<td>IOH = 400μA</td>
<td></td>
</tr>
<tr>
<td>IIL</td>
<td>Logical 0 Input Current P1, P2, P3</td>
<td>500</td>
<td>μA</td>
<td>Vin = 0.45V</td>
<td></td>
</tr>
<tr>
<td>IIL1</td>
<td>Logical 0 Input Current to EA/Vpp</td>
<td>-15</td>
<td>mA</td>
<td>Vin = 0.45V</td>
<td></td>
</tr>
<tr>
<td>IIL2</td>
<td>Logical 0 Input Current to XTAL2</td>
<td>-3.5</td>
<td>mA</td>
<td>XTAL1 = VSS, Vin = 0.45V</td>
<td></td>
</tr>
<tr>
<td>IL1</td>
<td>Input Leakage Current to Port 0</td>
<td>100</td>
<td>μA</td>
<td>0.45 &lt; Vin &lt; VCC</td>
<td></td>
</tr>
<tr>
<td>IIH</td>
<td>Logical Input Current to EA/VPP</td>
<td>500</td>
<td>μA</td>
<td>Vin = 2.4V</td>
<td></td>
</tr>
<tr>
<td>IIH1</td>
<td>Input Current to RST/VPP to activate reset</td>
<td>500</td>
<td>μA</td>
<td>Vin &lt; (VCC - 1.5V)</td>
<td></td>
</tr>
<tr>
<td>ICC</td>
<td>Power Supply Current</td>
<td>270</td>
<td>mA</td>
<td>All outputs disconnected, EA = VCC</td>
<td></td>
</tr>
<tr>
<td>CIO</td>
<td>Capacitance of I/O Buffers</td>
<td>10</td>
<td>pF</td>
<td>fc = 1 MHz, TA = 25°C</td>
<td></td>
</tr>
</tbody>
</table>
**AC CHARACTERISTICS** ($T_A = 0^\circ C$ to $70^\circ C$, $V_{CC} = 4.5$V to $5.5$V, $V_{SS} = 0$V; Load Capacitance for Port 0, ALE, and PSEN = 100 pF; Load Capacitance for all other outputs = 80 pF)

**EXTERNAL PROGRAM MEMORY CHARACTERISTICS**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8744H 12 MHz Osc</th>
<th>Variable Oscillator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>TCLCL</td>
<td>Oscillator Period</td>
<td>83.3</td>
<td>285.7</td>
</tr>
<tr>
<td>TLHLL</td>
<td>ALE Pulse Width</td>
<td>127</td>
<td>2TCLCL-40</td>
</tr>
<tr>
<td>TAVLL</td>
<td>Address Valid to ALE</td>
<td>53</td>
<td>TCLCL-40</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold after ALE</td>
<td>48</td>
<td>TCLCL-35</td>
</tr>
<tr>
<td>TLLIV</td>
<td>ALE to Valid Instr In</td>
<td>183</td>
<td>4TCLCL-150</td>
</tr>
<tr>
<td>TLLPL</td>
<td>ALE to PSEN</td>
<td>58</td>
<td>TCLCL-25</td>
</tr>
<tr>
<td>TPLPH</td>
<td>PSEN Pulse Width</td>
<td>190</td>
<td>3TCLCL-60</td>
</tr>
<tr>
<td>TPLIV</td>
<td>PSEN to Valid Instr In</td>
<td>100</td>
<td>3TCLCL-150</td>
</tr>
<tr>
<td>TPXIX</td>
<td>Input Instr Hold after PSEN</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TPXIZ</td>
<td>Input Instr Float after PSEN</td>
<td>63</td>
<td>TCLCL-20</td>
</tr>
<tr>
<td>TPXAV</td>
<td>PSEN to Address Valid</td>
<td>75</td>
<td>TCLCL-8</td>
</tr>
<tr>
<td>TAVIV</td>
<td>Address to Valid Instr In</td>
<td>267</td>
<td>5TCLCL-150</td>
</tr>
<tr>
<td>TAZPL</td>
<td>Address Float to PSEN</td>
<td>-25</td>
<td>-25</td>
</tr>
</tbody>
</table>

**EXTERNAL DATA MEMORY CHARACTERISTICS**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>8744H 12 MHz Osc</th>
<th>Variable Oscillator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>TRLRH</td>
<td>RD Pulse Width</td>
<td>400</td>
<td>6TCLCL-100</td>
</tr>
<tr>
<td>TWLWH</td>
<td>WR Pulse Width</td>
<td>400</td>
<td>6TCLCL-100</td>
</tr>
<tr>
<td>TLLAX</td>
<td>Address Hold After ALE</td>
<td>48</td>
<td>TCLCL-35</td>
</tr>
<tr>
<td>TRLDV</td>
<td>RD to Valid Data In</td>
<td>252</td>
<td>5TCLCL-165</td>
</tr>
<tr>
<td>TRHDX</td>
<td>Data Hold after RD</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TRHDZ</td>
<td>Data Float after RD</td>
<td>97</td>
<td>2TCLCL-70</td>
</tr>
<tr>
<td>TLLDV</td>
<td>ALE to Valid Data In</td>
<td>517</td>
<td>8TCLCL-150</td>
</tr>
<tr>
<td>TAVDV</td>
<td>Address to Valid Data In</td>
<td>585</td>
<td>9TCLCL-165</td>
</tr>
<tr>
<td>TLLWL</td>
<td>ALE to WR or RD</td>
<td>200</td>
<td>3TCLCL-50</td>
</tr>
<tr>
<td>TAVWL</td>
<td>Address to WR or RD</td>
<td>203</td>
<td>4TCLCL-130</td>
</tr>
<tr>
<td>TQVWX</td>
<td>Data Valid to WR Transition</td>
<td>13</td>
<td>TCLCL-70</td>
</tr>
<tr>
<td>TQVWH</td>
<td>Data Setup to WR High</td>
<td>433</td>
<td>7TCLCL-150</td>
</tr>
<tr>
<td>TWHQX</td>
<td>Data Held after WR</td>
<td>33</td>
<td>TCLCL-50</td>
</tr>
<tr>
<td>TRLAZ</td>
<td>RD Low to Address Float</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>TWHLH</td>
<td>RD or WR High to ALE High</td>
<td>33</td>
<td>TCLCL-50</td>
</tr>
</tbody>
</table>
AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS

AC testing inputs are driven at 2.4V for a logic 1 and 0.45V for a logic 0.
Timing measurements are made at 2.0V for a logic 1 and 0.8V for a logic 0.
For timing purposes, the float state is defined as the point at which a PO pin sinks 3.2mA or sources 400µA at the voltage test levels.

SERIAL I/O WAVEFORMS

Synchronous Data Transmission

Synchronous Data Reception

ORDER NUMBER: 210735-002
### Serial Interface

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDCY</td>
<td>Data Clock</td>
<td>420</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TDCL</td>
<td>Data Clock Low</td>
<td>180</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TDCH</td>
<td>Data Clock High</td>
<td>100</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TTD</td>
<td>Transmit Data Delay</td>
<td></td>
<td>180</td>
<td>ns</td>
</tr>
<tr>
<td>TDSS</td>
<td>Data Setup Time</td>
<td>40</td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>TDHS</td>
<td>Data Hold Time</td>
<td>40</td>
<td>ns</td>
<td></td>
</tr>
</tbody>
</table>

### Memory Access

#### Program Memory Read Cycle

```
<table>
<thead>
<tr>
<th>State</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALE</td>
<td>TLLPL</td>
</tr>
<tr>
<td>PSEN</td>
<td>TLLX</td>
</tr>
<tr>
<td>PORT 0</td>
<td>TAVL</td>
</tr>
<tr>
<td>PORT 2</td>
<td>ADDRESS</td>
</tr>
</tbody>
</table>
```

#### Data Memory Read Cycle

```
<table>
<thead>
<tr>
<th>State</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALE</td>
<td>TLLDV</td>
</tr>
<tr>
<td>PSEN</td>
<td>TLLWL</td>
</tr>
<tr>
<td>RD</td>
<td>TAVD</td>
</tr>
<tr>
<td>PORT 0</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>PORT 2</td>
<td>ADDRESS</td>
</tr>
</tbody>
</table>
```

#### Data Memory Write Cycle

```
<table>
<thead>
<tr>
<th>State</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALE</td>
<td>TWHLH</td>
</tr>
<tr>
<td>PSEN</td>
<td>TLLWL</td>
</tr>
<tr>
<td>WR</td>
<td>TAVW</td>
</tr>
<tr>
<td>PORT 0</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>PORT 2</td>
<td>ADDRESS</td>
</tr>
</tbody>
</table>
```
NOTE: VOL is degraded when the 8744H rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8744H as possible.

<table>
<thead>
<tr>
<th>Datum</th>
<th>Emitting Ports</th>
<th>Degraded I/O Lines</th>
<th>VOL (peak) (max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>P2, P0</td>
<td>P1, P3</td>
<td>0.8V</td>
</tr>
<tr>
<td>Write Data</td>
<td>P0</td>
<td>P1, P3, ALE</td>
<td>0.8V</td>
</tr>
</tbody>
</table>

**EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLCL</td>
<td>Oscillator Period: 8751H</td>
<td>83.3</td>
<td>285.7</td>
<td>ns</td>
</tr>
<tr>
<td>TCHCX</td>
<td>High Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCLCX</td>
<td>Low Time</td>
<td>20</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>TCLCH</td>
<td>Rise Time</td>
<td></td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td>TCHCL</td>
<td>Fall Time</td>
<td></td>
<td>20</td>
<td>ns</td>
</tr>
</tbody>
</table>
CLOCK WAVEFORMS

INTERNAL CLOCK

<table>
<thead>
<tr>
<th>STATE 4</th>
<th>STATE 5</th>
<th>STATE 6</th>
<th>STATE 1</th>
<th>STATE 2</th>
<th>STATE 3</th>
<th>STATE 4</th>
<th>STATE 5</th>
</tr>
</thead>
</table>

XTAL 2

ALE

EXTERNAL PROGRAM MEMORY FETCH

PSEN

P0

DATA SAMPLED

DATA SAMPLED

PCL OUT

PCL OUT

P2(EXT)

INDICATES ADDRESS TRANSITIONS

00H IS EMITTED DURING THIS PERIOD

PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)

READ CYCLE

RD

P0

DPL OR R1 OUT

DATA SAMPLED

P2

INDICATES DPH OR P2 SFR TO PCH TRANSITIONS

WRITE CYCLE

WR

P0

DPL OR R1 OUT

DATA OUT

P2

INDICATES DPH OR P2 SFR TO PCH TRANSITIONS

PORT OPERATION

MOV DEST, SRC

OLD DATA

NEW DATA

MOV DEST, P0

P0 PINS SAMPLED

MOV DEST, PORT (P1, P2, P3)

(INCLUDES INTO, INT1, T0, T1)

P0 PINS SAMPLED

SERIAL PORT SHIFT CLOCK

TXD (MODE 0)

RXD SAMPLED

RXD SAMPLED

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, \( T_A = 25^\circ C \), fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.
8744H EPROM CHARACTERISTICS

Erasure Characteristics
Erasure of the 8744H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Ångstroms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8744H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Ångstroms) to an integrated dose of at least 15 W-sec/cm² rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

Program Memory Security
The program memory security feature is developed around a “security bit” in the 8744H EPROM array. Once this “hidden bit” is programmed, electrical access to the contents of the entire program memory array becomes impossible. Activation of this feature is accomplished by programming the 8744H as described in “Programming the EPROM” with the exception that P2.6 is held at a TTL high rather than a TTL low. In addition, Port 1 and P2.0-P2.3 may be in any state. Figure 4 illustrates the security bit programming configuration. Reactivating the security feature, which again allows programmability of the EPROM, is accomplished by exposing the EPROM to ultraviolet light. This exposure, as described in “Erasure Characteristics,” erases the entire EPROM array. Therefore, attempted retrieval of “protected code” results in its destruction.

Program Verification
Program Memory may be read only when the “security feature” has not been activated. Refer to Figure 5 for Program Verification setup. To read the Program Memory, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4-P2.6 and PSEN should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for high.) EA/VPP is held normally high, and is pulsed to +21V. While EA/VPP is at 21V, the ALE/PROG pin, which is normally being held high, is pulsed low for 50 msec. Then EA/VPP is returned to high. This is illustrated in Figure 3. Detailed timing specifications are provided in the EPROM Programming and Verification Characteristics section of this data sheet.
Figure 3. Programming Configuration

Figure 4. Security Bit Programming Configuration
Figure 5. Program Verification Configuration

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION CHARACTERISTICS
(TA = 21°C to 27°C, VCC = 4.5V to 5.5V, VSS = 0V)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vpp</td>
<td>Programming Supply Voltage</td>
<td>20.5</td>
<td>21.5</td>
<td>V</td>
</tr>
<tr>
<td>IPP</td>
<td>Programming Current</td>
<td></td>
<td>30</td>
<td>mA</td>
</tr>
<tr>
<td>1/TCLCL</td>
<td>Oscillator Frequency</td>
<td></td>
<td>6</td>
<td>MHz</td>
</tr>
<tr>
<td>TAVGL</td>
<td>Address Setup to PROG</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGHAX</td>
<td>Address Hold after PROG</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDVGL</td>
<td>Data Setup to PROG</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TGHDX</td>
<td>Data Hold after PROG</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEHSH</td>
<td>ENABLE High to Vpp</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TSHGL</td>
<td>Vpp Setup to PROG</td>
<td>10</td>
<td></td>
<td>μsec</td>
</tr>
<tr>
<td>TGHSL</td>
<td>Vpp Hold after PROG</td>
<td>10</td>
<td></td>
<td>μsec</td>
</tr>
<tr>
<td>TGLGH</td>
<td>PROG Width</td>
<td>45</td>
<td>55</td>
<td>msec</td>
</tr>
<tr>
<td>TAVQV</td>
<td>Address to Data Valid</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TELQV</td>
<td>ENABLE to Data Valid</td>
<td>48TCLCL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEHQZ</td>
<td>Data Float after ENABLE</td>
<td>0</td>
<td>48TCLCL</td>
<td></td>
</tr>
</tbody>
</table>
EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION WAVEFORMS

For programming conditions see Figure 3.
For security bit programming conditions see Figure 4.
For verification conditions see Figure 5.
SUMMARY

The 8044 offers a lower cost and higher performance solution to networking microcontrollers than conventional solutions. The system cost is lowered by integrating an entire microcomputer with an intelligent HDLC/SDLC communication processor onto a single chip. The higher performance is realized by integrating two processors running concurrently on one chip; the powerful 8051 microcontroller and the Serial Interface Unit. The 8051 microcontroller is substantially off-loaded from the communication tasks when using the AUTO mode. In the AUTO mode the SIU handles many of the data link functions in hardware. The advantages of the AUTO mode are: less software is required to implement a secondary data link, the 8051 CPU is offloaded, and the turn-around time is reduced, thus increasing the network throughput. Currently the 8044 is the only microcontroller with a sophisticated communications processor on-chip. In the future there will be more microcontrollers available following this trend.

INTRODUCTION

Today microcontrollers are being designed into virtually every type of equipment. For the household, they are turning up in refrigerators, thermostats, burglar alarms, sprinklers, and even water softeners. At work they are found in laboratory instruments, copiers, elevators, hospital equipment, and telephones. In addition, a lot of microcomputer equipment contains more than one microcontroller. Applications using multiple microcontrollers as well, as the office and home, are now faced with the same requirements that laboratory instruments were faced with 12 years ago — they need to connect them together and have them communicate. This need was satisfied in the laboratory with the IEEE-488 General Purpose Instrumentation Bus (GPIB). However, GPIB does not meet the current design objectives for networking microcontrollers.

Today there are many communications schemes and protocols available; some of the popular ones are GPIB, Async, HDLC/SDLC, and Ethernet. Common design objectives of today’s networks are: low cost, reliable, efficient throughput, and expandable. In examining available solutions, GPIB does not meet these design objectives; first, the cable is too expensive (parallel communications), second, it can only be used over a limited distance (20 meters), and third, it can only handle a limited number of stations. For general networking, serial communications is preferable because of lower cable costs and higher reliability (fewer connections). While Ethernet provides very high performance, it is more of a system backbone rather than a microcontroller interconnect. Async, on the other hand, is inexpensive but it is not an efficient protocol for data block or file transfers. Even with some new modifications such as a 9 bit protocol for addressing, important functions such as

acknowledgements, error checking/recovery, and data transparency are not standardized nor supported by available data comm chips.

SDLC, Synchronous Data Link Control, meets the requirements for communications link design. The physical medium can be used on two or four wire twisted pair with inexpensive transceivers and connectors. It can also be interfaced through modems, which allows it to be used on broadband networks, leased or switched telephone lines. VLSI controllers have been available from a number of vendors for years; higher performance and more user friendly SDLC controllers continue to appear. SDLC has also been designed to be very reliable. A 16 bit CRC checks the integrity of the received data, while frame numbering and acknowledgements are also built in. Using SDLC, up to 254 stations can be uniquely addressed, while HDLC addressing is unlimited. If an RS-422 only requires a single +5 volt power supply.

What will the end user pay for the added value provided by communications? The cost of the communications hardware is not the only additional cost. There will be performance degradation in the main application because the microcontroller now has additional tasks to perform. There are two extremes to the cost of adding communication capability. One could spend very little by adding an I/O port and have the CPU handle everything from the baud rate to the protocol. Of course the main application would be idle while the CPU was communicating. The other extreme would be to add another microcontroller to the system dedicated to communications. This communications processor could interface to the main CPU through a high speed parallel link or dual port RAM. This approach would maintain system performance, but it would be costly.

Adding HDLC/SDLC Networking Capability

Figure 1 shows a microcomputer system with a conventional HDLC/SDLC communications solution. The additional hardware needed to realize the conventional design is: an HDLC/SDLC communication chip, additional ROM for the communication software, part of an interrupt controller, a baud rate generator, a phase locked loop, NRZI encoded/decoder, and a cable driver locked loop are used when the transmitter does not send the clock on a separate line from the data (i.e. over telephone lines, or two wire cable). The NRZI encoder/decoder is used in HDLC/SDLC to combine the clock into the data line. A phase locked loop is used to recover the clock from the data line.

The majority of the available communication chips provide a limited number of data link control functions. Most of them will handle Zero Bit Insertion/Deletion (ZBI/D), Flags, Aborts, Automatic
address recognition, and CRC generation and checking. It is the CPU’s responsibility to manage link access, command recognition and response, acknowledgements and error recovery. Handling these tasks can take a lot of CPU time. In addition, servicing the transmission and reception of data bytes can also be very time consuming depending on the method used.

Using a DMA controller can increase the overall system performance, since it can transfer a block of data in fewer clock cycles than a CPU. In addition, the CPU and the DMA controller can multiplex their access to the bus so that both can be running at virtually the same time. However, both the DMA controller and the CPU are sharing the same bus, therefore, neither one get to utilize 100% of the bus bandwidth. Microcontrollers available today do not support DMA, therefore, they would have to use interrupts, since polling is unacceptable in a multitasking environment.

In an interrupt driven, the CPU has overhead in addition to servicing the interrupt. During each interrupt request the CPU has to save all of the important registers, transfer a byte, update pointers and counters, then restore all of its registers. At low bit rates this overhead may be insignificant. However, the percentage of overhead increases linearly with the bit rate. At high bit rates this overhead would consume all of the CPU’s time. There is another nuisance factor associated with interrupt driven systems, interrupt latency. Too much interrupt latency will cause data to be lost from underrun and overrun errors.

The additional hardware necessary to implement the communications solution, as shown in Figure 1, would require 1 LSI chip and about 10 TTL chips. The cost of CPU throughput degradation can be even greater. The percentage of time the CPU has to spend servicing the communication tasks can be anywhere from 10-100%, depending on the serial bit rate. These high costs will prevent consumer acceptance of networking microcomputer equipment.

A Highly Integrated, High Performance Solution
The 8044 reduces the cost of networking microcontrollers without compromising performance. It contains all of the hardware components necessary to implement a microcomputer system with communications capability, plus it reduces the CPU and software overhead of implementing HDLC/SDLC. Figure 2 shows a functional block diagram of the 8044.

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Interface Unit to provide a single chip solution which efficiently implements a distributed processing or distributed control system. The microcontroller is a self sufficient unit containing ROM, RAM, ALU and its own peripherals. The 8044’s architecture and instruction set are identical to the 8051’s. The Serial Interface Unit (SIU) uses bit synchronous HDLC/SDLC protocol and can communicate at bit rates up to 2.4 Mbps, externally clocked, or up to 375 Kbps using the on-chip digital phase locked loop. The SIU contains its own processor, which operates concurrently with the microcontroller.

The CPU and the SIU, in the 8044, interface through 192 bytes of dual port RAM. There is no hardware arbitration in the dual port RAM. Both processor’s memory access cycles are interlaced; each processor has access every other clock cycle. Therefore, there
is no throughput loss in either processor as a result of the dual port RAM, and execution times are deterministic. Since this has always been the method for memory access on the 8051 microcontroller, 8051 programs have the same execution time in the 8044.

By integrating all of the communication hardware onto the 8051 microcontroller, the hardware cost of the system is reduced. Now several chips have been integrated into a single chip. This means that the system power is reduced, P.C. board space is reduced, inventory and assembly is reduced, and reliability is improved. The improvement in reliability is a result of fewer chips and interconnections on the P.C. board.

As mentioned before, there can be two extremes in a design which adds communications to the microcomputer system. The 8044 solution uses the high end extreme. The SIU on the 8044 contains its own processor which communicates with the 8051 processor through dual port RAM and control/status registers. While the SIU is not a totally independent communications processor, it substantially offloads the 8051 processor from the communication tasks.

The DMA on the 8044 is dedicated to the SIU. It cannot access external RAM. By having a DMA controller in the SIU, the 8051 CPU is offloaded. As a result of the dual port RAM design, the DMA does not share the running at full speed while the frames are being transmitted or received. Also, the nuisance of overrun and underrun errors is totally eliminated since the dedicated DMA controller is guaranteed to meet the maximum data rates. Having a dedicated DMA controller means that the serial channel interrupt can be the lowest priority, thus allowing the CPU to have higher priority real time interrupts.

Figure 3 shows a comparison between the conventional and the 8044 solution on the percentage of time the CPU must spend sending data. This diagram was derived by assuming a 64 byte information frame is repeatedly transmitted. The conventional solution is interrupt driven, and each interrupt service routine is assumed to take about 15 instructions with a 1 µsec instruction cycle time. At 533 Kbps, an interrupt would occur every 15 usec. Thus, the CPU becomes completely dedicated to servicing the serial communications. The conventional design could not support bit rates higher than this because of underruns and overrun. For the 8044 to repeatedly send 64 byte frames, it simply reinitializes the DMA controller. As a result, the 8044 can support bit rates up to 2.4 Mbps.

Some of the other communications tasks the CPU has to perform, such as link access, command recognition/response, and acknowledgements, are performed automatically by the SIU in a mode called "AUTO." The combination of the dedicated DMA controller and the AUTO mode, substantially offload
the CPU, thus allowing it to devote more of its power to other tasks.

8044's Auto Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC commands without CPU intervention. All AUTO mode responses to the primary station conform to IBM's SDLC definition. In the AUTO mode the 8044 can only be a secondary station operating in SDLC specified "Normal Response Mode." Normal Response Mode means that the secondary station can not transmit unless it is polled by the primary station. The SIU in the AUTO mode can recognize and respond to the following SDLC commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and for loop mode UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I, RR, and RNR. In addition, the SIU manages Ns and Nr in the control field. If it detects an error in either Ns or Nr, it interrupts the CPU for error recovery.

How does the SIU know what responses to send to the primary? It uses two status bits which are set by the CPU. The two bits are TBF (Transmit Buffer Full) and RBP (Receive Buffer Protect). TBF indicates that the CPU wants to send data, and RBP indicates that the receive data buffer is full. Table I shows the responses the SIU will send based on these two status bits. This is an innovative approach to communication design. The CPU in the 8044 with one instruction can directly set a bit which communicates to the primary what its transmit and receive buffering status is.

When the CPU wants to send a frame, it loads the transmit buffer with the data, loads the starting address and the count of the data into the SIU, then sets TBF to transmit the frame. The SIU waits for the primary station to poll it with a RR command. After the SIU is polled, it automatically sends the information frame to the primary with the proper control field. The SIU then waits for a positive acknowledgement from the primary before incrementing the Ns field and interrupting the CPU for more data. If a negative acknowledgement is received, the SIU automatically retransmits the frame.

When the 8044 is ready to receive information, the CPU loads the receive buffer starting address and the buffer length into the SIU, then enables the receiver. When a valid information frame with the correct address and CRC is received, the SIU will increment the Nr field, disable the receiver and interrupt the CPU indicating that a good frame has been received. The CPU then sets RBP, reenables the receiver and processes the received data. By enabling the receiver with RBP set, the SIU will automatically respond to polls with a Receive Not Ready, thus keeping the link moving rather than timing out the primary from a disabled receiver, or interrupting the CPU with another poll before it has processed the data. After the data has been processed, the CPU clears RBP, returning to the Receive Ready responses.
Table 1. SIU's automatic responses in auto mode

<table>
<thead>
<tr>
<th>STATUS BITS</th>
<th>RESPONSE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TBF</strong></td>
<td><strong>RBP</strong></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

SDLC communications can be broken up into four states: Logical Disconnect State, Initialization State, Frame Reject State, and Information Transfer State. Data can only be transferred in the Information Transfer State. More than 90% of the time a station will be in the Information Transfer State, which is where the SIU can run autonomously. In the other states, where error recovery, online/offline, and initialization takes place, the CPU manages the protocol.

In the Information Transfer State there are three common events which occur as illustrated in Figure 4, they are: 1) the primary polls the secondary and the secondary is ready to receive but has nothing to send, 2) the primary sends the secondary information, and 3) the secondary sends information to the primary. Figures 5, 6, and 7 compare the functions the conventional design and the 8044 must execute in order to respond to the primary for the cases in Figure 4.

![Figure 4. SDLC commands and responses in the information transfer state](image-url)
### CASE 1

**8044 AUTO MODE**

<table>
<thead>
<tr>
<th>PRIMARY</th>
<th>CONVENTIONAL DESIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR Poll</td>
<td>Receive interrupts</td>
</tr>
<tr>
<td></td>
<td>Decode received control field</td>
</tr>
<tr>
<td></td>
<td>Check NR field</td>
</tr>
<tr>
<td></td>
<td>Load response into transmit control field</td>
</tr>
<tr>
<td></td>
<td>Send frame</td>
</tr>
<tr>
<td></td>
<td>Transmit interrupts</td>
</tr>
</tbody>
</table>

Figure 5. Primary polls secondary, secondary has nothing to send

### CASE 2

**8044 AUTO MODE**

<table>
<thead>
<tr>
<th>PRIMARY</th>
<th>CONVENTIONAL DESIGN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load transmit buffer</td>
<td>Load transmit buffer and transmit control byte</td>
</tr>
<tr>
<td>Set TBF bit</td>
<td>Receive interrupts</td>
</tr>
<tr>
<td>RR poll</td>
<td>Decode receive control byte</td>
</tr>
<tr>
<td></td>
<td>Check NR field</td>
</tr>
<tr>
<td></td>
<td>Send frame</td>
</tr>
<tr>
<td></td>
<td>Transmit interrupts</td>
</tr>
<tr>
<td></td>
<td>Receive interrupts</td>
</tr>
<tr>
<td></td>
<td>Decode receive control byte</td>
</tr>
<tr>
<td></td>
<td>Check NR field</td>
</tr>
<tr>
<td></td>
<td>Increment NS</td>
</tr>
<tr>
<td>RR poll</td>
<td></td>
</tr>
<tr>
<td>Transmit interrupt</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Primary polls secondary, secondary sends information frame
CASE 3

8044 AUTO MODE PRIMARY

AR-3Q7

CONVENTIONAL DESIGN

Receive interrupts
Decode received control field
Check NR field
Load response into transmit control field
Send frame
Transmit interrupts

Receive interrupts
Decode receive control field
Check NS NR fields
Increment NR
Load response into transmit control field
Send frame
Transmit interrupts

Receive interrupt

Receive interrupt
I frame

Figure 7. Primary sends information frame to secondary

Using case 1 as an example, the conventional design first gets receive interrupts bringing the data from the SDLC comm chip into memory. The CPU must then decode the command in the control field and determine the response. In addition, it must check the NR field for any pending acknowledgements. The CPU loads the transmit buffer with the appropriate address and control field, then transmits the frame. When the 8044 receives this frame in AUTO mode, the CPU never gets an interrupt because the SIU handles the entire frame reception and response automatically.

In SDLC networks, when there is no information transfers, case 1 is the activity on the line. Typically this is 80% of the network traffic. The CPU in the conventional design would constantly be getting interrupts and servicing the communications tasks, even when it has nothing to send or receive. On the other hand, the 8044 CPU would only get involved in communicating when it has data to send or receive.

Having the SIU implement a subset of the SDLC protocol in hardware not only offloads the CPU, but it also improves the throughput on the network. The most critical parameter for calculating throughput on any high speed network is the station turnaround time; the time it takes a station to respond after receiving a frame. Since the 8044 handles all of the commands and responses of the Information Transfer State in hardware, the turnaround time is much faster than handling it in software, hence a higher throughput.

8044's Flexible Mode

In the "NON-AUTO" mode or Flexible mode, the SIU does not recognize or respond to any commands, nor does it manage acknowledgements, which means the CPU must handle link access, command recognition/response, acknowledgements and error recovery by itself. The Flexible mode allows the 8044 to have extended address fields and extended control fields, thus providing HDLC support. In the Flexible mode the 8044 can operate as a primary station, since it can transmit without being polled.

Front End Communications Processor

The 8044 can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capable of extensively off-loading link control functions for...
the microprocessor. In some applications the 8044 can even be used for communications preprocessing, in addition to data link control. For this type of design the 8044 would communicate to the Host CPU through a FIFO, or dual port RAM. A block diagram of this design is given in Figure 8. A tightly coupled interface between the 8044 and the Host CPU would be established. The Host CPU would give the 8044 high level commands and data which the 8044 would convert to HDLC/SDLC. This particular type of design would be most appropriate for a primary Station which is normally a micro, mini, or mainframe computer. Sophisticated secondary stations could also take advantage of this design.

Since the 8044 has ROM on chip, all the communications software is non-volatile. The 8044 primary station could down-line-load software to 8044 secondary stations. Once down-line-loading is implemented, software updates to the primary and secondary stations could be done very inexpensively. The only things which would remain fixed in ROM are the HDLC/SDLC communications software and the software interface to the HOST.

![Diagram of 8044 front end processor](image)

Figure 8. 8044 front end processor
Designing Microcontroller Systems for Electrically Noisy Environments

Contents

SYMPTOMS OF NOISE PROBLEMS ........ 22-2

TYPES AND SOURCES OF ELECTRICAL NOISE
Supply Line Transients ............... 22-2
EMP and RFI .......................... 22-2
ESD .................................. 22-3
Ground Noise .......................... 22-3

"RADIATED" AND "CONDUCTED" NOISE ... 22-3

SIMULATING THE ENVIRONMENT ........ 22-4

TYPES OF FAILURES AND FAILURE MECHANISMS .......... 22-4

THE GAME PLAN .......................... 22-5

CURRENT LOOPS ...................... 22-5

SHIELDING .............................. 22-6
Shielding Against Capacitive Coupling .... 22-6
Shielding Against Inductive Coupling ... 22-6
RF Shielding ......................... 22-9

GROUNDS ...................... 22-10
Safety Ground ......................... 22-10
Signal Ground .......................... 22-11
Practical Grounding .................... 22-12
Braided Cable ......................... 22-13

POWER SUPPLY DISTRIBUTION AND DECOUPLING ........ 22-14
Selecting the Value of the Decoupling Cap .... 22-15
The Case for On-Board Voltage Regulation .... 22-16

RECOVERING GRACEFULLY FROM A SOFTWARE UPSET 22-16

SPECIAL PROBLEM AREAS ................. 22-18
ESD .................................. 22-18
The Automotive Environment .............. 22-19

PARTING THOUGHTS ..................... 22-21

REFERENCES ........................ 22-22
Digital circuits are often thought of as being immune to noise problems, but really they're not. Noises in digital systems produce software upsets: program jumps to apparently random locations in memory. Noise-induced glitches in the signal lines can cause such problems, but the supply voltage is more sensitive to glitches than the signal lines.

Severe noise conditions, those involving electrostatic discharges, or as found in automotive environments, can do permanent damage to the hardware. Electrostatic discharges can blow a crater in the silicon. In the automotive environment, in ordinary operation, the “12V” power line can show + and −400V transients.

This Application Note describes some electrical noises and noise environments. Design considerations, along the lines of PCB layout, power supply distribution and decoupling, and shielding and grounding techniques, that may help minimize noise susceptibility are reviewed. Special attention is given to the automotive and ESD environments.

Symptoms of Noise Problems

Noise problems are not usually encountered during the development phase of a microcontroller system. This is because benches rarely simulate the system’s intended environment. Noise problems tend not to show up until the system is installed and operating in its intended environment. Then, after a few minutes or hours of normal operation the system finds itself somewhere out in left field. Inputs are ignored and outputs are gibberish. The system may respond to a reset, or it may have to be turned off physically and then back on again, at which point it commences operating as though nothing had happened. There may be an obvious cause, such as an electrostatic discharge from somebody’s finger to a keyboard or the upset occurs every time a copier machine is turned on or off. Or there may be no obvious cause, and nothing the operator can do will make the upset repeat itself. But a few minutes, or a few hours, or a few days later it happens again.

One symptom of electrical noise problems is randomness, both in the occurrence of the problem and in what the system does in its failure. All operational upsets that occur at seemingly random intervals are not necessarily caused by noise in the system. Marginal VCC, inadequate decoupling, rarely encountered software conditions, or timing coincidences can produce upsets that seem to occur randomly. On the other hand, some noise sources can produce upsets downright periodically. Nevertheless, the more difficult it is to characterize an upset as to cause and effect, the more likely it is to be a noise problem.

Types and Sources of Electrical Noise

The name given to electrical noises other than those that are inherent in the circuit components (such as thermal noise) is EMI: electromagnetic interference. Motors, power switches, fluorescent lights, electrostatic discharges, etc., are sources of EMI. There is a veritable alphabet soup of EMI types, and these are briefly described below.

SUPPLY LINE TRANSIENTS

Anything that switches heavy current loads onto or off of AC or DC power lines will cause large transients in these power lines. Switching an electric typewriter on or off, for example, can put a 1000V spike onto the AC power lines.

The basic mechanism behind supply line transients is shown in Figure 1. The battery represents any power source, AC or DC. The coils represent the line inductance between the power source and the switchable loads R1 and R2. If both loads are drawing current, the line current flowing through the line inductance establishes a magnetic field of some value. Then, when one of the loads is switched off, the field due to that component of the line current collapses, generating transient voltages, \( v = L \frac{di}{dt} \), which try to maintain the current at its original level. That’s called an “inductive kick.” Because of contact bounce, transients are generated whether the switch is being opened or closed, but they’re worse when the switch is being opened.

An inductive kick of one type or another is involved in most line transients, including those found in the automotive environment. Other mechanisms for line transients exist, involving noise pickup on the lines. The noise voltages are then conducted to a susceptible circuit right along with the power.

EMP AND RFI

Anything that produces arcs or sparks will radiate electromagnetic pulses (EMP) or radio-frequency interference (RFI).

![Figure 1. Supply Line Transients](AFN-02131A)
Spark discharges have probably caused more software upsets in digital equipment than any other single noise source. The upsetting mechanism is the EMP produced by the spark. The EMP induces transients in the circuit, which are what actually cause the upset.

Arcs and sparks occur in automotive ignition systems, electric motors, switches, static discharges, etc. Electric motors that have commutator bars produce an arc as the brushes pass from one bar to the next. DC motors and the "universal" (AC/DC) motors that are used to power hand tools are the kinds that have commutator bars. In switches, the same inductive kick that puts transients on the supply lines will cause an opening or closing switch to throw a spark.

**ESD**

Electrostatic discharge (ESD) is the spark that occurs when a person picks up a static charge from walking across a carpet, and then discharges it into a keyboard, or whatever else can be touched. Walking across a carpet in a dry climate, a person can accumulate a static voltage of 35kV. The current pulse from an electrostatic discharge has an extremely fast risetime—typically, 4A/nsec. Figure 2 shows ESD waveforms that have been observed by some investigators of ESD phenomena.

It is enlightening to calculate the \( L(\frac{di}{dt}) \) voltage required to drive an ESD current pulse through a couple of inches of straight wire. Two inches of straight wire has about 50nH of inductance. That's not very much, but using 50nH for \( L \) and 4A/nsec for \( \frac{di}{dt} \) gives an \( L(\frac{di}{dt}) \) drop of about 200V. Recent observations by W.M. King suggest even faster risetimes (Figure 2B) and the occurrence of multiple discharges during a single discharge event.

Obviously, ESD-sensitivity needs to be considered in the design of equipment that is going to be subjected to it, such as office equipment.

**GROUND NOISE**

Currents in ground lines are another source of noise. These can be 60Hz currents from the power lines, or RF hash, or crosstalk from other signals that are sharing this particular wire as a signal return line. Noise in the ground lines is often referred to as a "ground loop" problem. The basic concept of the ground loop is shown in Figure 3. The problem is that true earth-ground is not really at the same potential in all locations. If the two ends of a wire are earth-grounded at different locations, the voltage difference between the two "ground" points can drive significant currents (several amperes) through the wire. Consider the wire to be part of a loop which contains, in addition to the wire, a voltage source that represents the difference in potential between the two ground points, and you have the classical "ground loop." By extension, the term is used to refer to any unwanted (and often unexpected) currents in a ground line.

"**Radiated**" and "**Conducted**" Noise

Radiated noise is noise that arrives at the victim circuit in the form of electromagnetic radiation, such as EMP and RFI. It causes trouble by inducing extraneous voltages in the circuit. Conducted noise is noise that arrives at the victim circuit already in the form of an extraneous voltage, typically via the AC or DC power lines.

One defends against radiated noise by care in designing layouts and the use of effective shielding techniques. One defends against conducted noise with filters and suppres-

![Figure 2. Waveforms of Electrostatic Discharge Currents From a Hand-Held Metallic Object](AFN-02131A)
Sors, although layouts and grounding techniques are important here, too.

Simulating the Environment

Addressing noise problems after the design of a system has been completed is an expensive proposition. The ill will generated by failures in the field is not cheap either. It's cheaper in the long run to invest a little time and money in learning about noise and noise simulation equipment, so that controlled tests can be made on the bench as the design is developing.

Simulating the intended noise environment is a two-step process: First you have to recognize what the noise environment is, that is, you have to know what kinds of electrical noises are present, and which of them are going to cause trouble. Don't ignore this first step, because it's important. If you invest in an induction coil spark generator just because your application is automotive, you'll be straining at the gnat and swallowing the camel. Spark plug noise is the least of your worries in that environment.

The second step is to generate the electrical noise in a controlled manner. This is usually more difficult than first imagined; one first imagines the simulation in terms of a waveform generator and a few spare parts, and then finds that a wideband power amplifier with a 200V dynamic range is also required. A good source of information on who supplies what noise-simulating equipment is the 1981 "ITEM" Directory and Design Guide (reference 6).

Types of Failures and Failure Mechanisms

A major problem that EMI can cause in digital systems is intermittent operational malfunction. These software upsets occur when the system is in operation at the time an EMI source is activated, and are usually characterized by a loss of information or a jump in the execution of the program to some random location in memory. The person who has to iron out such problems is tempted to say the program counter went crazy. There is usually no damage to the hardware, and normal operation can resume as soon as the EMI has passed or the source is de-activated. Resuming normal operation usually requires manual or automatic reset, and possibly re-entering of lost information.

Electrostatic discharges from operating personnel can cause not only software upsets, but also permanent ("hard") damage to the system. For this to happen the system doesn't even have to be in operation. Sometimes the permanent damage is latent, meaning the initial damage may be marginal and require further aggravation through operating stress and time before permanent failure takes place. Sometimes too the damage is hidden.

One ESD-related failure mechanism that has been identified has to do with the bias voltage on the substrate of the chip. On some CPU chips the substrate is held at -2.5V by a phase-shift oscillator working into a capacitor/diode clamping circuit. This is called a "charge pump" in chip-design circles. If the substrate wanders too far in either direction, program read errors are noted. Some designs have been known to allow electrostatic discharge currents to flow directly into port pins of an 8048. The resulting damage to the oxide causes an increase in leakage current, which loads down the charge pump, reducing the substrate voltage to a marginal or unacceptable level. The system is then unreliable or completely inoperative until the CPU chip is replaced. But if the CPU chip was subjected to a discharge spark once, it will eventually happen again.

Chips that have a grounded substrate, such as the 8748, can sometimes sustain some oxide damage without actually becoming inoperative. In this case the damage is present, and the increased leakage current is noted; however, since the substrate voltage retains its design value, the damage is largely hidden.

![Figure 3. What a Ground Loop Is](image-url)
It must therefore be recognized that connecting port pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges, makes an extremely dangerous configuration. It doesn't make any difference what CPU chip is being used, or who makes it. If it connects unprotected to a keyboard, it will eventually be destroyed. Designing for an ESD-environment will be discussed further on.

We might note here that MOS chips are not the only components that are susceptible to permanent ESD damage. Bipolar and linear chips can also be damaged in this way. PN junctions are subject to a hard failure mechanism called thermal secondary breakdown, in which a current spike, such as from an electrostatic discharge, causes microscopically localized spots in the junction to approach melt temperatures. Low power TTL chips are subject to this type of damage, as are op-amps. Op-amps, in addition, often carry on-chip MOS capacitors which are directly across an external pin combination, and these are susceptible to dielectric breakdown.

We return now to the subject of software upsets. Noise transients can upset the chip through any pin, even an output pin, because every pin on the chip connects to the substrate through a pn junction. However, the most vulnerable pin is probably the VCC line, since it has direct access to all parts of the chip: every register, gate, flip-flop and buffer.

The menu of possible upset mechanisms is quite lengthy. A transient on the substrate at the wrong time will generally cause a program read error. A false level at a control input can cause an extraneous or misdirected opcode fetch. A disturbance on the supply line can flip a bit in the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the power-up reset capacitor to discharge. Thus when the transient ends, the chip starts up again without a reset.

A common failure mode is for the processor to lock itself into a tight loop. Here it may be executing the data in a table, or the program counter may have jumped a notch, so that the processor is now executing operands instead of opcodes, or it may be trying to fetch opcodes from a nonexistent external program memory.

It should be emphasized that mechanisms for upsets have to do with the arrival of noise-induced transients at the pins of the chips, rather than with the generation of noise pulses within the chip itself, that is, it's not the chip that is picking up noise, it's the circuit.

The Game Plan

Prevention is usually cheaper than suppression, so first we'll consider some preventive methods that might help to minimize the generation of noise voltages in the circuit. These methods involve grounding, shielding, and wiring techniques that are directed toward the mechanisms by which noise voltages are generated in the circuit. We'll also discuss methods of decoupling. Then we'll look at some schemes for making a graceful recovery from upsets that occur in spite of preventive measures. Lastly, we'll take another look at two special problem areas: electrostatic discharges and the automotive environment.

Current Loops

The first thing most people learn about electricity is that current won't flow unless it can flow in a closed loop. This simple fact is sometimes temporarily forgotten by the overworked engineer who has spent the past several years mastering the intricacies of the DO loop, the timing loop, the feedback loop, and maybe even the ground loop. The simple current loop probably owes its apparent demise to the invention of the ground symbol. By a stroke of the pen one avoids having to draw the return paths of most of the current loops in the circuit. Then "ground" turns into an infinite current sink, so that any current that flows into it is gone and forgotten. Forgotten it may be, but it's not gone. It must return to its source, so that its path will by all the laws of nature form a closed loop.

The physical geometry of a given current loop is the key to why it generates EMI, why it's susceptible to EMI, and how to shield it. Specifically, it's the area of the loop that matters.

Any flow of current generates a magnetic field whose intensity varies inversely to the distance from the wire that carries the current. Two parallel wires conducting currents +I and -I (as in signal feed and return lines) would generate a nonzero magnetic field near the wires, where the distance from a given point to one wire is noticeably different from the distance to the other wire, but farther away (relative to the wire spacing), where the distances from a given point to either wire are about the same, the fields from both wires tend to cancel out. Thus, maintaining proximity between feed and return paths is an important way to minimize their interference with other signals. The way to maintain their proximity is essentially to minimize their loop area. And, because the mutual inductance from current loop A to current loop B is the same as the mutual inductance from current loop B to current loop A, a circuit that doesn't radiate interference doesn't receive it either.

Thus, from the standpoint of reducing both generation of EMI and susceptibility to EMI, the hard rule is to keep loop areas small. To say that loop areas should be minimized is the same as saying the circuit inductance should
be minimized. Inductance is by definition the constant of proportionality between current and the magnetic field it produces: \( L = \frac{d}{dt} \). Holding the feed and return wires close together so as to promote field cancellation can be described either as minimizing the loop area or as minimizing L. It's the same thing.

**Shielding**

There are three basic kinds of shields: shielding against capacitive coupling, shielding against inductive coupling, and RF shielding. Capacitive coupling is electric field coupling, so shielding against it amounts to shielding against electric fields. As will be seen, this is relatively easy. Inductive coupling is magnetic field coupling, so shielding against it is shielding against magnetic fields. This is a little more difficult. Strangely enough, this type of shielding does not in general involve the use of magnetic materials. RF shielding, the classical "metallic barrier" against all sorts of electromagnetic fields, is what most people picture when they think about shielding. Its effectiveness depends partly on the selection of the shielding material, but mostly, as it turns out, on the treatment of its seams and the geometry of its openings.

**SHIELDING AGAINST CAPACITIVE COUPLING**

Capacitive coupling involves the passage of interfering signals through mutual or stray capacitances that aren't shown on the circuit diagram, but which the experienced engineer knows are there. Capacitive coupling to one's body is what would cause an unstable oscillator to change its frequency when the person reaches his hand over the circuit, for example. More importantly, in a digital system it causes crosstalk in multi-wire cables.

The way to block capacitive coupling is to enclose the circuit or conductor you want to protect in a metal shield. That's called an electrostatic or Faraday shield. If coverage is 100%, the shield does not have to be grounded, but it usually is, to ensure that circuit-to-shield capacitances go to signal reference ground rather than act as feedback and crosstalk elements. Besides, from a mechanical point of view, grounding it is almost inevitable.

A grounded Faraday shield can be used to break capacitive coupling between a noisy circuit and a victim circuit, as shown in Figure 4. Figure 4A shows two circuits capacitively coupled through the stray capacitance between them. In Figure 4B the stray capacitance is intercepted by a grounded Faraday shield, so that interference currents are shunted to ground. For example, a grounded plane can be inserted between PCBs (printed circuit boards) to eliminate most of the capacitive coupling between them.

Another application of the Faraday shield is in the electrostatically shielded transformer. Here, a conducting foil is laid between the primary and secondary coils so as to intercept the capacitive coupling between them. If a system is being upset by AC line transients, this type of transformer may provide the fix. To be effective in this application, the shield must be connected to the green-wire ground.

**SHIELDING AGAINST INDUCTIVE COUPLING**

With inductive coupling, the physical mechanism involved is a magnetic flux density \( B \) from some external interference source that links with a current loop in the victim circuit, and generates a voltage in the loop in accordance with Lenz's law: \( v = -NA(dB/dt) \), where in this case \( N = 1 \) and \( A \) is the area of the current loop in the victim circuit.

There are two aspects to defending a circuit against inductive pickup. One aspect is to try to minimize the offensive fields at their source. This is done by minimizing the area of the current loop at the source so as to promote field cancellation, as described in the section on current loops. The other aspect is to minimize the inductive pickup in the victim circuit by minimizing the area of that current loop, since, from Lenz's law, the induced voltage is proportional to this area. So the two aspects really involve the same corrective action: minimize the areas of the current loops. In other words, minimizing the offensiveness of a circuit inherently minimizes its susceptibility.

![Figure 4. Use of Faraday Shield](AFN-02131A)
Shielding against inductive coupling means nothing more nor less than controlling the dimensions of the current loops in the circuit. We must look at four examples of this type of "shielding": the coaxial cable, the twisted pair, the ground plane, and the gridded-ground PCB layout.

**The Coaxial Cable** — Figure 5 shows a coaxial cable carrying a current I from a signal source to a receiving load. The shield carries the same current as the center conductor. Outside the shield, the magnetic field produced by +I flowing in the center conductor is cancelled by the field produced by -I flowing in the shield. To the extent that the cable is ideal in producing zero external magnetic field, it is immune to inductive pickup from external sources. The cable adds effectively zero area to the loop. This is true only if the shield carries the same current as the center conductor.

In the real world, both the signal source and the receiving load are likely to have one end connected to a common signal ground. In that case, should the cable be grounded at one end, both ends, or neither end? The answer is that it should be grounded at both ends. Figure 6A shows the situation when the cable shield is grounded at only one end. In that case the current loop runs down the center conductor of the cable, then back through the common ground connection. The loop area is not well defined. The shield not only does not carry the same current as the center conductor, but it doesn't carry any current at all. There is no field cancellation at all. The shield has no effect whatsoever on either the generation of EMI or susceptibility to EMI. (It is, however, still effective as an electrostatic shield, or at least it would be if the shield coverage were 100%.)

Figure 6B shows the situation when the cable is grounded at both ends. Does the shield carry all of the return current, or only a portion of it on account of the shunting effect of the common ground connection? The answer to that question depends on the frequency content of the signal. In general, the current loop will follow the path of least impedance. At low frequencies, 0Hz to several kHz, where the inductive reactance is insignificant, the current will follow the path of least resistance. Above a few kHz, where inductive reactance predominates, the current will follow the path of least inductance. The path of least inductance is the path of minimum loop area. Hence, for higher frequencies the shield carries virtually the same current as the center conductor, and is therefore effective against both generation and reception of EMI.

Note that we have now introduced the famous "ground loop" problem, as shown in Figure 7A. Fortunately, a digital system has some built-in immunity to moderate ground loop noise. In a noisy environment, however, one can break the ground loop, and still maintain the shielding effectiveness of the coaxial cable, by inserting an optical coupler, as shown in Figure 7B. What the optical coupler does, basically, is allow us to re-define the signal source as being ungrounded, so that that end of the cable need not be grounded, and still lets the shield carry the same current as the center conductor. Obviously, if the signal source weren't grounded in the first place, the optical coupler wouldn't be needed.

**The Twisted Pair** — A cheaper way to minimize loop area is to run the feed and return wires right next to each other. This isn't as effective as a coaxial cable in minimizing loop area. An ideal coaxial cable adds zero area to the loop, whereas merely keeping the feed and return wires next to each other is bound to add a finite area.

However, two things work to make this cheaper method almost as good as a coaxial cable. First, real coaxial cables are not ideal. If the shield current isn't evenly distributed around the center conductor at every cross-
section of the cable (it isn't), then field cancellation external to the shield is incomplete. If field cancellation is incomplete, then the effective area added to the loop by the cable isn't zero. Second, in the cheaper method the feed and return wires can be twisted together. This not only maintains their proximity, but the noise picked up in one twist tends to cancel out the noise picked up in the next twist down the line. Thus the "twisted pair" turns out to be about as good a shield against inductive coupling as coaxial cable is.

The twisted pair does not, however, provide electrostatic shielding (i.e., shielding against capacitive coupling). Another operational difference between them is that the coaxial cable works better at higher frequencies. This is primarily because the twisted pair adds more capacitive loading to the signal source than the coaxial cable does. The twisted pair is normally considered useful up to only about 1MHz, as opposed to near a GHz for the coaxial cable.

The Ground Plane — The best way to minimize loop areas when many current loops are involved is to use a ground plane. A ground plane is a conducting surface that is to serve as a return conductor for all the current loops in the circuit. Normally, it would be one or more layers of a multilayer PCB. All ground points in the circuit go not to a grounded trace on the PCB, but directly to the ground plane. This leaves each current loop in the circuit free to complete itself in whatever configuration yields minimum loop area (for frequencies wherein the ground path impedance is primarily inductive).

Thus, if the feed path for a given signal zigzags its way across the PCB, the return path for this signal is free to zigzag right along beneath it on the ground plane, in such a configuration as to minimize the energy stored in the magnetic field produced by this current loop. Minimal magnetic flux means minimal effective loop area and minimal susceptibility to inductive coupling.

The Gridded-Ground PCB Layout — The next best thing to a ground plane is to layout the ground traces on a PCB in the form of a grid structure, as shown in Figure 8. Laying horizontal traces on one side of the board and vertical traces on the other side allows the passage of signal and power traces. Wherever vertical and horizontal ground traces cross, they must be connected by a feed-through.

Have we not created here a network of "ground loops"? Yes, in the literal sense of the word, but loops in the ground layout on a PCB are not to be feared. Such inoffensive little loops have never caused as much noise pickup as their avoidance has. Trying to avoid innocent little loops in the ground layout, PCB designers have forced current loops into geometries that could swallow a whale. That is exactly the wrong thing to do.

The gridded ground structure works almost as well as the ground plane, as far as minimizing loop area is concerned. For a given current loop, the primary return path may have to zig once in a while where its feed path zags,
but you still get a mathematically optimal distribution of currents in the grid structure, such that the current loop produces less magnetic flux than if the return path were restrained to follow any single given ground trace. The key to attaining minimum loop areas for all the current loops together is to let the ground currents distribute themselves around the entire area of the board as freely as possible. They want to minimize their own magnetic field. Just let them.

**RF SHIELDING**

A time-varying electric field generates a time-varying magnetic field, and vice versa. Far from the source of a time-varying EM field, the ratio of the amplitudes of the electric and magnetic fields is always 377 ohms. Up close to the source of the fields, however, this ratio can be quite different, and dependent on the nature of the source. Where the ratio is near 377 ohms is called the far field, and where the ratio is significantly different from 377 ohms is called the near field. The ratio itself is called the wave impedance, E/H.

The near field goes out about 1/6 of a wavelength from the source. At 1 MHz this is about 150 feet, and at 10 MHz it's about 15 feet. That means if an EMI source is in the same room with the victim circuit, it's likely to be a near field problem. The reason this matters is that in the near field an RF interference problem could be almost entirely due to E-field coupling or H-field coupling, and that could influence the choice of an RF shield or whether an RF shield will help at all.

In the near field of a whip antenna, the E/H ratio is higher than 377 ohms, which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically the same as inductive coupling. Methods to protect a circuit from inductive coupling, such as a gridded-ground structure, would be effective against RF interference from a loop antenna. A Faraday shield would be less effective.

More formally, the PR losses are referred to as absorption loss, and the re-radiation is called reflection loss. As it turns out, absorption loss is the primary shielding mechanism for H-fields, and reflection loss is the primary shielding mechanism for E-fields. Reflection loss, being a surface phenomenon, is pretty much independent of the thickness of the shielding material. Both loss mechanisms, however, are dependent on the frequency (ω) of the impinging EMI field, and on the permeability (μ) and conductivity (σ) of the shielding material. These loss mechanisms vary approximately as follows:

reflection loss to an E-field (in dB) \( \sim \log \frac{\sigma}{\omega \mu} \)

absorption loss to an H-field (in dB) \( \sim \frac{1}{\sqrt{\omega \mu}} \)

where \( t \) is the thickness of the shielding material.

The first expression indicates that E-field shielding is more effective if the shield material is highly conductive, and less effective if the shield is ferromagnetic, and that low-frequency fields are easier to block than high-frequency fields. This is shown in Figure 9.

In the near field of a loop antenna, the E/H ratio is lower than 377 ohms, which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically capacitive coupling. Methods to protect a circuit from capacitive coupling, such as a Faraday shield, would be effective against RF interference from a whip antenna. A gridded-ground structure would be less effective.
Copper and aluminum both have the same permeability, but copper is slightly more conductive, and so provides slightly greater reflection loss to an E-field. Steel is less effective for two reasons. First, it has a somewhat elevated permeability due to its iron content, and, second, as tends to be the case with magnetic materials, it is less conductive.

On the other hand, according to the expression for absorption loss to an H-field, H-field shielding is more effective at higher frequencies and with shield material that has both high conductivity and high permeability. In practice, however, selecting steel for its high permeability involves some compromise in conductivity. But the increase in permeability more than makes up for the decrease in conductivity, as can be seen in Figure 10. This figure also shows the effect of shield thickness.

A composite of E-field and H-field shielding is shown in Figure 11. However, this type of data is meaningful only in the far field. In the near field the EMI could be 90% H-field, in which case the reflection loss is irrelevant. It would be advisable then to beef up the absorption loss, at the expense of reflection loss, by choosing steel. A better conductor than steel might be less expensive, but quite ineffective.

A different shielding mechanism that can be taken advantage of for low frequency magnetic fields is the ability of a high permeability material such as mumetal to divert the field by presenting a very low reluctance path to the magnetic flux. Above a few kHz, however, the permeability of such materials is the same as steel.

In actual fact the selection of a shielding material turns out to be less important than the presence of seams, joints and holes in the physical structure of the enclosure. The shielding mechanisms are related to the induction of currents in the shield material, but the currents must be allowed to flow freely. If they have to detour around slots and holes, as shown in Figure 12, the shield loses much of its effectiveness.

As can be seen in Figure 12, the severity of the detour has less to do with the area of the hole than it does with the geometry of the hole. Comparing Figure 12C with 12D shows that a long narrow discontinuity such as a seam can cause more RF leakage than a line of holes with larger total area. A person who is responsible for designing or selecting rack or chassis enclosures for an EMI environment needs to be familiar with the techniques that are available for maintaining electrical continuity across seams. Information on these techniques is available in the references.

### Grounds

There are two kinds of grounds: earth-ground and signal ground. The earth is not an equipotential surface, so earth ground potential varies. That and its other electrical properties, are not conducive to its use as a return conductor in a circuit. However, circuits are often connected to earth ground for protection against shock hazards. The other kind of ground, signal ground, is an arbitrarily selected reference node in a circuit—the node with respect to which other node voltages in the circuit are measured.

#### SAFETY GROUND

The standard 3-wire single-phase AC power distribution system is represented in Figure 13. The white wire is earth-grounded at the service entrance. If a load circuit has a metal enclosure or chassis, and if the black wire develops a short to the enclosure, there will be a shock hazard to operating personnel, unless the enclosure itself is earth-grounded. If the enclosure is earth-grounded, a
short results in a blown fuse rather than a “hot” enclosure. The earth-ground connection to the enclosure is called a safety ground. The advantage of the 3-wire power system is that it distributes a safety ground along with the power.

Note that the safety-ground wire carries no current, except in case of a fault, so that at least for low frequencies it’s at earth-ground potential along its entire length. The white wire, on the other hand, may be several volts off ground, due to the IR drop along its length.

**SIGNAL GROUND**

Signal ground is a single point in a circuit that is designated to be the reference node for the circuit. Commonly, wires that connect to this single point are also referred to as “signal ground.” In some circles “power supply common” or PSC is the preferred terminology for these conductors. In any case, the manner in which these wires connect to the actual reference point is the basis of distinction among three kinds of signal-ground wiring methods: series, parallel, and multipoint. These methods are shown in Figure 14.

The series connection is pretty common because it’s simple and economical. It’s the noisiest of the three, however, due to common ground impedance coupling between the circuits. When several circuits share a ground wire, currents from one circuit, flowing through the finite impedance of the common ground line, cause variations in the ground potential of the other circuits. Given that the currents in a digital system tend to be spiked, and that the common impedance is mainly inductive reactance, the variations could be bad enough to cause bit errors in high current or particularly noisy situations.

The parallel connection eliminates common ground impedance problems, but uses a lot of wire. Other disadvantages are that the impedance of the individual ground lines can be very high, and the ground lines themselves can become sources of EMI.
In the multipoint system, ground impedance is minimized by using a ground plane with the various circuits connected to it by very short ground leads. This type of connection would be used mainly in RF circuits above 10MHz.

**PRACTICAL GROUNDING**

A combination of series and parallel ground-wiring methods can be used to trade off economic and the various electrical considerations. The idea is to run series connections for circuits that have similar noise properties, and connect them at a single reference point, as in the parallel method, as shown in Figure 15.

In Figure 15, “noisy signal ground” connects to things like motors and relays. Hardware ground is the safety ground connection to chassis, racks, and cabinets. It’s a mistake to use the hardware ground as a return path for signal currents because it’s fairly noisy (for example, it’s the hardware ground that receives an ESD spark) and tends to have high resistance due to joints and seams.

![Figure 14. Three Ways to Wire the Grounds](image)

![Figure 15. Parallel Connection of Series Grounds](image)

Screws and bolts don’t always make good electrical connections because of galvanic action, corrosion, and dirt. These kinds of connections may work well at first, and then cause mysterious maladies as the system ages.

Figure 16 illustrates a grounding system for a 9-track digital tape recorder, showing an application of the series/parallel ground-wiring method.

Figure 17 shows a similar separation of grounds at the PCB level. Currents in multiplexed LED displays tend to put a lot of noise on the ground and supply lines because of the constant switching and changing involved in the scanning process. The segment driver ground is relatively quiet, since it doesn’t conduct the LED currents. The digit driver ground is noisier, and should be provided with a separate path to the PCB ground terminal, even if the PCB ground layout is gridded. The LED feed and return current paths should be laid out on opposite sides of the board like parallel flat conductors.

Figure 18 shows right and wrong ways to make ground connections in racks. Note that the safety ground connections from panel to rack are made through ground straps, not panel screws. Rack 1 correctly connects signal ground to rack ground only at the single reference point. Rack 2 incorrectly connects signal ground to rack ground at two points, creating a ground loop around points 1, 2, 3, 4, 1.

Breaking the “electronics ground” connection to point 1 eliminates the ground loop, but leaves signal ground in rack 2 sharing a ground impedance with the relatively noisy hardware ground to the reference point; in fact, it may end up using hardware ground as a return path for signal and power supply currents. This will probably cause more problems than the ground loop.

**BRAIDED CABLE**

Ground impedance problems can be virtually eliminated by using braided cable. The reduction in impedance is due to skin effect: At higher frequencies the current tends to flow along the surface of a conductor rather than uni-
Figure 16. Ground System in a 9-Track Digital Recorder

Figure 17. Separate Ground for Multiplexed LED Display
formly through its bulk. While this effect tends to increase the impedance of a given conductor, it also indicates the way to minimize impedance, and that is to manipulate the shape of the cross-section so as to provide more surface area. For its bulk, braided cable is almost pure surface.

**Power Supply Distribution and Decoupling**

The main consideration for power supply distribution lines is, as for signal lines, to minimize the areas of the current loops. But the power supply lines take on an importance that no signal line has when one considers the fact that these lines have access to every PC board in the system. The very extensiveness of the supply current loops makes it difficult to keep loop areas small. And, a noise glitch on a supply line is a glitch delivered to every board in the system.

The power supply provides low-frequency current to the load, but the inductance of the board-to-board and chip-to-chip distribution network makes it difficult for the power supply to maintain VCC specs on the chip while providing the current spikes that a digital system requires. In addition, the power supply current loop is a very large one, which means there will be a lot of noise pick-up. Figure 19A shows a load circuit trying to draw current spikes from a supply voltage through the line impedance. To the VCC waveform shown in that figure should be added the inductive pick-up associated with a large loop area.

Adding a decoupling capacitor solves two problems: The capacitor acts as a nearby source of charge to supply the current spikes through a smaller line impedance, and it defines a much smaller loop area for the higher frequency components of EMI. This is illustrated in Figure 19B, which shows the capacitor supplying the current spike, during which VCC drops from 5V by the amount indicated in the figure. Between current spikes the capacitor recovers through the line impedance.

One should resist the temptation to add a resistor or an inductor to the decoupler so as to form a genuine RC or LC low-pass filter because that slows down the speed with which the decoupler cap can be refreshed. Good filtering and good decoupling are not necessarily the same thing.

The current loop for the higher frequency currents, then, is defined by the decoupling cap and the load circuit, rather than by the power supply and the load circuit. For the decoupling cap to be able to provide the current spikes required by the load, the inductance of this current loop must be kept small, which is the same as saying the loop area must be kept small. This is also the requirement for minimizing inductive pick-up in the loop.

There are two kinds of decoupling caps: board decouplers and chip decouplers. A board decoupler will normally be a 10 to 100μf electrolytic capacitor placed near to where the power supply enters the PC board, but its placement is relatively non-critical. The purpose of the board decoupler is to refresh the charge on the chip decouplers. The chip decouplers are what actually provide the current spikes to the chips. A chip decoupler will normally be a 0.1 to 1μf ceramic capacitor placed near the chip and connected to the chip by traces that minimize the area of the loop formed by the cap and the chip. If a chip decoupler is not properly placed on the board, it will be ineffective as a decoupler and will serve only to increase

---

**Figure 18. Electronic Circuits Mounted in Equipment Racks Should Have Separate Ground Connections. Rack 1 Shows Correct Grounding, Rack 2 Shows Incorrect Grounding**

---

AFN-02131A
the cost of the board. Good and bad placement of decoupling capacitors are illustrated in Figure 20.

Power distribution traces on the PC board need to be laid out so as to obtain minimal area (minimal inductance) in the loops formed by each chip and its decoupler, and by the chip decouplers and the board decoupler. One way to accomplish this goal is to use a power plane. A power plane is the same as a ground plane, but at VCC potential. More economically, a power grid similar to the ground grid previously discussed (Figure 8) can be used. Actually, if the chip decoupling loops are small, other aspects of the power layout are less critical. In other words, power planes and power gridding aren’t needed, but power traces should be laid in the closest possible proximity to ground traces, preferably so that each power trace is on the direct opposite side of the board from a ground trace.

Special-purpose power supply distribution buses which mount on the PCB are available. The buses use a parallel flat conductor configuration, one conductor being a VCC line and the other a ground line. Used in conjunction with a gridded ground layout, they not only provide a low-inductance distribution system, but can themselves form part of the ground grid, thus facilitating the PCB layout. The buses are available with and without enhanced bus capacitance, under the names Mini/Bus® and Q/PAC® from Rogers Corp. (5750 E. McKellips, Mesa, AZ 85205).

SELECTING THE VALUE OF THE DECOUPLING CAP

The effectiveness of the decoupling cap has a lot to do with the way the power and ground traces connect this capacitor to the chip. In fact, the area formed by this loop is more important than the value of the capacitance. Then, given that the area of this loop is indeed minimal, it can generally be said that the larger the value of the decoupling cap, the more effective it is, if the cap has a mica, ceramic, glass, or polystyrene dielectric.

It’s often said, and not altogether accurately, that the chip decoupler shouldn’t have too large a value. There are two reasons for this statement. One is that some capacitors, because of the nature of their dielectrics, tend to become inductive or lossy at higher frequencies. This is true of
electrolytic capacitors, but mica, glass, ceramic, and poly-
Styrene dielectrics work well to several hundred MHz. The other reason cited for not using too large a capaci-
tance has to do with lead inductance. The capacitor with its lead inductance forms a series LC circuit. Below the frequency of series resonance, the net impedance is inductive. Thus a decou-
pling capacitor is capacitive only below the frequency of series resonance. This frequency is given by

\[ f_0 = \frac{1}{2\pi\sqrt{LC}} \]

where \( C \) is the decoupling capacitance and \( L \) is the lead inductance between the capacitor and the chip. On a PC board this inductance is determined by the layout, and is the same whether the capacitor dropped into the PCB holes is 0.001\( \mu \)F or 1\( \mu \)F. Thus, increasing the capacitance lowers the series resonant frequency. In fact, according to the resonant frequency formula, increasing \( C \) by a factor of 100 lowers the resonant frequency by a factor of 10.

Figures quoted on the series resonant frequency of a 0.01\( \mu \)F capacitor run from 10 to 15MHz, depending on the lead length. If these numbers were accurate, a 1\( \mu \)F capacitor in the same position on the board would have a resonant frequency of 1.0 to 1.5MHz, and as a decoupler would do more harm than good. However, the numbers are based on a presumed inductance of a given length of wire (the lead length). It should be noted that a "length of wire" has no inductance at all, strictly speaking. Only a complete current loop has inductance, and the inductance depends on the geometry of the loop. Figures quoted on the inductance of a length of wire are based on a presumably "very large" loop area, such that the magnetic field produced by the return current has no cancellation effect on the field produced by the current in the given length of wire. Such a loop geometry is not and should not be the case with the decoupling loop.

Figure 21 shows VCC waveforms, measured between pins 40 and 20 (VCC and VSS) of an 8751 CPU, for several conditions of decoupling on a PC board that has a decoupling loop area slightly larger than necessary. These photographs show the effects of increasing the decoupling capacitance and decreasing the area of the decoupling loop. The indications are that a 1\( \mu \)F capacitor is better than a 0.1\( \mu \)F capacitor, which in turn is better than nothing, and that the board should have been laid out with more attention paid to the area of the decoupling loop.

Figure 21E was obtained using a special-purpose experimental capacitor designed by Rogers Corp. 'Q-Pac Division, Mesa, AZ) for use as a decoupler. It consists of two parallel plates, the length of a 40-pin DIP, separated by a ceramic dielectric. Sandwiched between the CPU chip and the PCB (or between the CPU socket and the PCB), it makes connection to pins 40 and 20, forming a leadless decoupling capacitor. It is obviously a configuration of minimal inductance. Unfortunately, the particular sample tested had only 0.07\( \mu \)F of capacitance and so was unable to prevent the 1MHz ripple as effectively as the configuration of Figure 21D. It seems apparent, though, that with more capacitance this part will alleviate a lot of decoupling problems.

THE CASE FOR ON-BOARD VOLTAGE REGULATION

To complicate matters, supply line glitches aren't always picked up in the distribution networks, but can come from the power supply circuit itself. In that case, a well-designed distribution network faithfully delivers the glitch throughout the system. The VCC glitch in Figure 22 was found to be coming from within a bench power supply in response to the EMP produced by an induction coil spark generator that was being used at Intel during a study of noise sensitivity. The VCC glitch is about 400mV high and some 20\( \mu \)sec in duration. Normal board decoupling techniques were ineffective in removing it, but adding an on-board voltage regulator chip did the job.

Thus, a good case can be made in favor of using a voltage regulator chip on each PCB, instead of doing all the voltage regulation at the supply circuit. This eases requirements on the heat-sinking at the supply circuit, and alleviates much of the distribution and board decoupling headaches. However, it also brings in the possibility that different boards would be operating at slightly different VCC levels due to tolerance in the regulator chips; this then leads to slightly different logic levels from board to board. The implications of that may vary from nothing to latch-up, depending on what kinds of chips are on the boards, and how they react to an input "high" that is perhaps 0.4V higher than local VCC.

Recovering Gracefully from a Software Upset

Even when one follows all the best guidelines for design-

ing for a noisy environment, it's always possible for a

noise transient to occur which exceeds the circuit's immunity level. In that case, one can strive at least for a graceful recovery.

Graceful recovery schemes involve additional hardware and/or software which is supposed to return the system to a normal operating mode after a software upset has occurred. Two decisions have to be made: How to recognize when an upset has occurred, and what to do about it.

If the designer knows what kinds and combinations of
No Decoupling Cap

0.1µf Decoupler Stretched Directly from Pin 40 to Pin 20, under the Socket. (This prevents the 1MHz ripple, but there's no reduction in higher frequency components. Further increases in capacitance effected no further improvement.)

Special-Purpose Decoupling Cap under Development by Rogers Corp. (Further discussion in text.)

Figure 21. Noise on VCC Line
outputs can legally be generated by the system, he can use gates to recognize and flag the occurrence of an illegal state of affairs. The flag can then trigger a jump to a recovery routine which then may check or re-initialize data, perhaps output an error message, or generate a simple reset.

The most reliable scheme is to use a so-called watchdog circuit. Here the CPU is programmed to generate a periodic signal as long as the system is executing instructions in an expected manner. The periodic signal is then used to hold off a circuit that will trigger a jump to a recovery routine. The periodic signal needs to be AC-coupled to the trigger circuit so that a "stuck-at" fault won't continue to hold off the trigger. Then, if the processor locks up someplace, the periodic signal is lost and the watchdog triggers a reset.

In practice, it may be convenient to drive the watchdog circuit with a signal which is being generated anyway by the system. One needs to be careful, however, that an upset does in fact discontinue that signal. Specifically, for example, one could use one of the digit drive signals going to a multiplexed display. But display scanning is often handled in response to a timer-interrupt, which may continue operating even though the main program is in a failure mode. Even so, with a little extra software, the signal can be used to control the watchdog (see reference 8 on this).

Simpler schemes can work well for simpler systems. For example, if a CPU isn't doing anything but scanning and decoding a keyboard, there's little to lose and much to gain by simply resetting it periodically with an astable multivibrator. It only takes about 13µsec (at 6MHz) to reset an 8048 if the clock oscillator is already running.

A zero-cost measure is simply to fill all unused program memory with NOPs and JMPs. It's still possible, of course, to get hung up in a data table or something. But you get a lot of protection, for the cost.

Further discussion of graceful recovery schemes can be found in reference 13.

Special Problem Areas

ESD

MOS chips have some built-in protection against a static charge build-up on the pins, as would occur during normal handling, but there's no protection against the kinds of current levels and rise times that occur in a genuine electrostatic spark. These kinds of discharges can blow a crater in the silicon.

It must be recognized that connecting CPU pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges makes an extremely fragile configuration. Buffering them is the very least one can do. But buffering doesn't completely solve the problem, because then the buffer chips will sustain the damage (even TTL); therefore, one might consider mounting the buffer chips in sockets for ease of replacement.

Transient suppressors, such as the TranZorbs® made by General Semiconductor Industries (Tempe, AZ), may in the long run provide the cheapest protection if their "zero inductance" structure is used. The structure and circuit application are shown in Figure 23.

The suppressor element is a p-n junction that operates like a Zener diode. Back-to-back units are available for AC operation. The element is more or less an open circuit at normal system voltage (the standoff voltage rating for the device), and conducts like a Zener diode at the clamping voltage.

The lead inductance in the conventional transient suppressor package makes the conventional package essen-

Figure 22. EMP-Induced Glitch
tially useless for protection against ESD pulses, owing to the fast rise of these pulses. The "zero inductance" units are available singly in a 4-pin DIP, and in arrays of four to a 16-pin DIP for PCB level protection. In that application they should be mounted in close proximity to the chips they protect.

In addition, metal enclosures or frames or parts that can receive an ESD spark should be connected by braided cable to the green-wire ground. Because of the ground impedance, ESD current shouldn't be allowed to flow through any signal ground, even if the chips are protected by transient suppressors. A 35kV ESD spark can always spare a few hundred volts to drive a fast current pulse down a signal ground line if it can't find a braided cable to follow. Think how delighted your 8048 will be to find its VSS pin about 250V higher than VCC for a few 10s of nanoseconds.

THE AUTOMOTIVE ENVIRONMENT
The automobile presents an extremely hostile environment for electronic systems. There are several parts to it:

1. Temperature extremes from -40°C to +125°C (under the hood) or +85°C (in the passenger compartment)
2. Electromagnetic pulses from the ignition system
3. Supply line transients that will knock your socks off

One needs to take a long, careful look at the temperature extremes. The allowable storage temperature range for most Intel MOS chips is -65°C to +150°C, although some chips have a maximum storage temperature rating of +125°C. In operation (or "under bias," as the data sheets say) the allowable ambient temperature range depends on the product grade, as follows:

<table>
<thead>
<tr>
<th>Grade</th>
<th>Ambient Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min.</td>
</tr>
<tr>
<td>Commercial</td>
<td>0</td>
</tr>
<tr>
<td>Industrial</td>
<td>-40</td>
</tr>
<tr>
<td>Automotive</td>
<td>-40</td>
</tr>
<tr>
<td>Military</td>
<td>-55</td>
</tr>
</tbody>
</table>

The different product grades are actually the same chip, but tested according to different standards. Thus, a given commercial-grade chip might actually pass military temperature requirements, but not have been tested for it. (Of course, there are other differences in grading requirements having to do with packaging, burn-in, traceability, etc.)

In any case, it's apparent that commercial-grade chips can't be used safely in automotive applications, not even in the passenger compartment. Industrial-grade chips can be used in the passenger compartment, and automotive or military chips are required in under-the-hood applications.

Ignition noise, CB radios, and that sort of thing are probably the least of your worries. In a poorly designed system, or in one that has not been adequately tested for the automotive environment, this type of EMI might cause a few software upsets, but not destroy chips.

The major problem, and the one that seems to come as the biggest surprise to most people, is the line transients. Regrettably, the 12V battery is not actually the source of power when the car is running. The charging system is, and it's not very clean. The only time the battery is the real source of power is when the car is first being started, and in that condition the battery terminals may be delivering about 5 or 6V. Below is a brief description of the major idiosyncracies of the "12V" automotive power line.
An abrupt reduction in the alternator load causes a positive voltage transient called "load dump." In a load dump transient the line voltage rises to 20 or 30V in a few msec, then decays exponentially with a time constant of about 100msec, as shown in Figure 24. Much higher peak voltages and longer decay times have also been reported. The worst case load dump is caused by disconnecting a low battery from the alternator circuit while the alternator is running. Normally this would happen intermittently when the battery terminal connections are defective.

When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40 and -100V for 100 msec or more.

Miscellaneous solenoid switching transients, such as the one shown in Figure 25, can drive the line to + or -200 to 400V for several μsec.

Mutual coupling between unshielded wires in long harnesses can induce 100 and 200V transients in unprotected circuits.

What all this adds up to is that people in the business of building systems for automotive applications need a comprehensive testing program. An SAE guideline which describes the automotive environment is available to designers: SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," 1980 SAE Handbook, Part 1, pp. 22.80-22.96.

Some suggestions for protecting circuitry are shown in Figure 26. A transient suppressor is placed in front of the regulator chip to protect it. Since the rise times in these transients are not like those in ESD pulses, lead inductance is less critical and conventional devices can be used. The regulator itself is pretty much of a necessity, since a load dump transient is simply not going to be removed...
by any conventional LC or RC filter. Special I/O interfacing is also required, because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are being monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in reference 2.

**Parting Thoughts**

The main sources of information for this Application Note were the references by Ott and by White. Reference 5 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society
EMC Education Committee
345 East 47th Street
New York, NY 10017
Phone: (212) 752-6800

Don White Consultants, Inc.
International Training Centre
P.O. Box D
Gainesville, VA 22065
Phone: (703) 347-0030

The EMC Education committee has available a video tape: “Introduction to EMC — A Video Training Tape,” by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.
June 1983

Oscillators for Microcontrollers

Tom Williamson
Microcontroller
Technical Marketing

Order Number: 230659-001
**CONTENTS**

**Oscillators for Microcontrollers**

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTRODUCTION</strong></td>
<td>22-25</td>
</tr>
<tr>
<td><strong>FEEDBACK OSCILLATORS</strong></td>
<td></td>
</tr>
<tr>
<td>Loop Gain</td>
<td>22-25</td>
</tr>
<tr>
<td>How Feedback Oscillators Work</td>
<td>22-26</td>
</tr>
<tr>
<td>The Positive Reactance Oscillator</td>
<td>22-26</td>
</tr>
<tr>
<td><strong>QUARTZ CRYSTALS</strong></td>
<td></td>
</tr>
<tr>
<td>Crystal Parameters</td>
<td>22-27</td>
</tr>
<tr>
<td>equivalent circuit</td>
<td>22-27</td>
</tr>
<tr>
<td>load capacitance</td>
<td>22-27</td>
</tr>
<tr>
<td>“series” vs. “parallel” crystals</td>
<td>22-28</td>
</tr>
<tr>
<td>equivalent series resistance</td>
<td>22-28</td>
</tr>
<tr>
<td>frequency tolerance</td>
<td>22-28</td>
</tr>
<tr>
<td>drive level</td>
<td>22-29</td>
</tr>
<tr>
<td><strong>CERAMIC RESONATORS</strong></td>
<td>22-29</td>
</tr>
<tr>
<td>Specifications for Ceramic Resonators</td>
<td>22-30</td>
</tr>
<tr>
<td><strong>OSCILLATOR DESIGN CONSIDERATIONS</strong></td>
<td>22-30</td>
</tr>
<tr>
<td>On-Chip Oscillators</td>
<td>22-30</td>
</tr>
<tr>
<td>crystal specifications</td>
<td>22-30</td>
</tr>
<tr>
<td>oscillation frequency</td>
<td>22-30</td>
</tr>
<tr>
<td>selection of CX1 and CX2</td>
<td>22-31</td>
</tr>
<tr>
<td>placement of components</td>
<td>22-31</td>
</tr>
<tr>
<td>clocking other chips</td>
<td>22-31</td>
</tr>
<tr>
<td>External Oscillators</td>
<td>22-31</td>
</tr>
<tr>
<td>gate oscillators vs. discrete devices</td>
<td>22-33</td>
</tr>
<tr>
<td>fundamental vs. overtone operation</td>
<td>22-33</td>
</tr>
<tr>
<td>“series” vs. “parallel” operation</td>
<td>22-33</td>
</tr>
<tr>
<td><strong>MORE ABOUT USING THE “ON-CHIP” OSCILLATORS</strong></td>
<td>22-34</td>
</tr>
<tr>
<td>Oscillator Calculations</td>
<td>22-34</td>
</tr>
<tr>
<td>Start-Up Characteristics</td>
<td>22-35</td>
</tr>
<tr>
<td>Steady-State Characteristics</td>
<td>22-37</td>
</tr>
<tr>
<td>Pin Capacitance</td>
<td>22-38</td>
</tr>
<tr>
<td>MCS®-51 Oscillator</td>
<td>22-39</td>
</tr>
<tr>
<td>MCS®-48 Oscillator</td>
<td>22-39</td>
</tr>
<tr>
<td>Pre-Production Tests</td>
<td>22-42</td>
</tr>
<tr>
<td>troubleshooting oscillator problems</td>
<td>22-43</td>
</tr>
<tr>
<td><strong>APPENDIX I</strong></td>
<td></td>
</tr>
<tr>
<td>Quartz and Ceramic Resonator Formulas</td>
<td>22-46</td>
</tr>
<tr>
<td><strong>APPENDIX II</strong></td>
<td></td>
</tr>
<tr>
<td>Oscillator Analysis Program</td>
<td>22-48</td>
</tr>
</tbody>
</table>
INTRODUCTION

Intel's microcontroller families (MCS®-48, MCS-51, and iACX-96) contain a circuit that is commonly referred to as the "on-chip oscillator." The on-chip circuitry is not itself an oscillator, of course, but an amplifier that is suitable for use as the amplifier part of a feedback oscillator. The data sheets and Microcontroller Handbook show how the on-chip amplifier and several off-chip components can be used to design a working oscillator. With proper selection of off-chip components, these oscillator circuits will perform better than almost any other type of clock oscillator, and by almost any criterion of excellence. The suggested circuits are simple, economical, stable, and reliable.

We offer assistance to our customers in selecting suitable off-chip components to work with the on-chip oscillator circuitry. It should be noted, however, that Intel cannot assume the responsibility of writing specifications for the off-chip components of the complete oscillator circuit, nor of guaranteeing the performance of the finished design in production, anymore than a transistor manufacturer, whose data sheets show a number of suggested circuits, can assume responsibility for the operation, in production, of any of them.

We are often asked why we don't publish a list of required crystal or ceramic resonator specifications, and recommend values for the other off-chip components. This has been done in the past, but sometimes with consequences that were not intended.

Suppose we suggest a maximum crystal resistance of 30 ohms for some given frequency. Then your crystal supplier tells you the 30-ohm crystals are going to cost twice as much as 50-ohm crystals. Fearing that Intel will not "guarantee operation" with 50-ohm crystals, you order the expensive ones. In fact, Intel guarantees only what is embodied within an Intel product. Besides, there is no reason why 50-ohm crystals couldn't be used, if the other off-chip components are suitably adjusted.

Should we recommend values for the other off-chip components? Should we do it for 50-ohm crystals or 30-ohm crystals? With respect to what should we optimize their selection? Should we minimize start-up time or maximize frequency stability? In many applications, neither start-up time nor frequency stability are particularly critical, and our "recommendations" are only restricting your system to unnecessary tolerances. It all depends on the application.

Although we will neither "specify" nor "recommend" specific off-chip components, we do offer assistance in these tasks. Intel applications engineers are available to provide whatever technical assistance may be needed or desired by our customers in designing with Intel products.

This Application Note is intended to provide such assistance in the design of oscillator circuits for microcontroller systems. Its purpose is to describe in a practical manner how oscillators work, how crystals and ceramic resonators work (and thus how to spec them), and what the on-chip amplifier looks like electronically and what its operating characteristics are. A BASIC program is provided in Appendix II to assist the designer in determining the effects of changing individual parameters. Suggestions are provided for establishing a pre-production test program.

FEEDBACK OSCILLATORS

Loop Gain

Figure 1 shows an amplifier whose output line goes into some passive network. If the input signal to the amplifier is \( v_1 \), then the output signal from the amplifier is \( v_2 = A v_1 \) and the output signal from the passive network is \( v_3 = \beta v_2 = \beta A v_1 \). Thus \( \beta A \) is the overall gain from terminal 1 to terminal 3.

![Figure 1 — Factors in Loop Gain](image)

Now connect terminal 1 to terminal 3, so that the signal path forms a loop: 1 to 2 to 3, which is also 1. Now we have a feedback loop, and the gain factor \( \beta A \) is called the loop gain.

Gain factors are complex numbers. That means they have a magnitude and a phase angle, both of which vary with frequency. When writing a complex number, one must specify both quantities, magnitude and angle. A number whose magnitude is 3, and whose angle is 45 degrees is commonly written this way: \( 3/45^\circ \). The number 1 is, in complex number notation, \( 1/0^\circ \), while \(-1\) is \( 1/180^\circ \).

By closing the feedback loop in Figure 0, we force the equality

\[ v_1 = \beta A v_1 \]

This equation has two solutions:

1) \( v_1 = 0 \);

2) \( \beta A = 1/0^\circ \).

In a given circuit, either or both of the above solutions may be in effect. In the first solution the circuit is quiescent (no output signal). If you're trying to make an
How Feedback Oscillators Work

A feedback oscillator amplifies its own noise and feeds it back to itself in exactly the right phase, at the oscillation frequency, to build up and reinforce the desired oscillations. Its ability to do that depends on its loop gain. First, oscillations can occur only at the frequency for which the loop gain has a phase angle of 0 degrees. Second, build-up of oscillations will occur only if the loop gain exceeds 1 at that frequency. Build-up continues until nonlinearities in the circuit reduce the average value of the loop gain to exactly 1.

Start-up characteristics depend on the small-signal properties of the circuit, specifically, the small-signal loop gain. Steady-state characteristics of the oscillator depend on the large-signal properties of the circuit, such as the transfer curve (output voltage vs. input voltage) of the amplifier, and the clamping effect of the input protection devices. These things will be discussed more fully further on. First we will look at the basic operation of a particular oscillator circuit, called the "positive reactance" oscillator.

The Positive Reactance Oscillator

Figure 2 shows the configuration of the positive reactance oscillator. The inverting amplifier, working into the impedance of the feedback network, produces an output signal that is nominally 180 degrees out of phase with its input. The feedback network must provide an additional 180 degrees phase shift, such that the overall loop gain has zero (or 360) degrees phase shift at the oscillation frequency.

\[
X_f = \frac{+1}{\omega C}
\]

where \(X_f\) is the reactance of \(Z_f\) (the total \(Z_f\) being \(R_f + jX_f\)), and \(C\) is the series combination of \(C_{X1}\) and \(C_{X2}\):

\[
C = \frac{C_{X1}C_{X2}}{C_{X1} + C_{X2}}
\]

In other words, \(Z_f\) and \(C\) form a parallel resonant circuit.

If \(Z_f\) is an inductor, then \(X_f = \omega L\), and the frequency at which the loop gain has zero phase is the frequency at which

\[
\omega L = \frac{1}{\omega C}
\]

or

\[
\omega = \frac{1}{\sqrt{LC}}
\]

Normally, \(Z_f\) is not an inductor, but it must still have a positive reactance in order for the circuit to oscillate. There are some piezoelectric devices on the market that show a positive reactance, and provide a more stable oscillation frequency than an inductor will. Quartz crystals can be used where the oscillation frequency is critical, and lower cost ceramic resonators can be used where the frequency is less critical.

When the feedback element is a piezoelectric device, this circuit configuration is called a Pierce oscillator. The advantage of piezoelectric resonators lies in their property of providing a wide range of positive reactance values over a very narrow range of frequencies. The reactance will equal \(1/\omega C\) at some frequency within this range, so the oscillation frequency will be within the same range. Typically, the width of this range is only .3% of the nominal frequency of a quartz crystal, and about 3% of the nominal frequency of a ceramic resonator. With relatively little design effort, frequency accuracies of .03% or better can be obtained with quartz crystals, and .3% or better with ceramic resonators.

QUARTZ CRYSTALS

The crystal resonator is a thin slice of quartz sandwiched between two electrodes. Electrically, the device looks pretty much like a 5 or 6 pF capacitor, except that over certain ranges of frequencies the crystal has a positive (i.e., inductive) reactance.

The ranges of positive reactance originate in the piezoelectric property of quartz: Squeezing the crystal gener-
ates an internal E-field. The effect is reversible: Applying an E-field causes a mechanical deflection. Applying an AC E-field causes the crystal to vibrate. At certain vibrational frequencies there is a mechanical resonance. As the E-field frequency approaches a frequency of mechanical resonance, the measured reactance of the crystal becomes positive, as shown in Figure 3.

Typically there are several ranges of frequencies wherein the reactance of the crystal is positive. Each range corresponds to a different mode of vibration in the crystal. The main resonances are the so-called fundamental response and the third and fifth overtone responses.

The overtone responses shouldn't be confused with the harmonics of the fundamental. They're not harmonics, but different vibrational modes. They're not in general at exact integer multiples of the fundamental frequency. There will also be "spurious" responses, occurring typically a few hundred KHz above each main response.

To assure that an oscillator starts in the desired mode on power-up, something must be done to suppress the loop gain in the undesired frequency ranges. The crystal itself provides some protection against unwanted modes of oscillation; too much resistance in that mode, for example. Additionally, junction capacitances in the amplifying devices tend to reduce the gain at higher frequencies, and thus may discriminate against unwanted modes. In some cases a circuit fix is necessary, such as inserting a trap, a phase shifter, or ferrite beads to kill oscillations in unwanted modes.

**Crystal Parameters**

**Equivalent Circuit**

Figure 4 shows an equivalent circuit that is used to represent the crystal for circuit analysis.

The \( R_1 L_1 C_1 \) branch is called the motional arm of the crystal. The values of these parameters derive from the mechanical properties of the crystal and are constant for a given mode of vibration. Typical values for various nominal frequencies are shown in Table 1.

\[
f_s = \frac{1}{2\pi \sqrt{L_1 C_1}}
\]

At this frequency the impedance of the crystal is \( R_1 \) in parallel with the reactance of \( C_0 \). For most purposes, this impedance is taken to be just \( R_1 \), since the reactance of \( C_0 \) is so much larger than \( R_1 \).

**Load Capacitance**

A crystal oscillator circuit such as the one shown in Figure 2 (redrawn in Figure 5) operates at the frequency for which the crystal is antiresonant (ie; parallel-resonant) with the total capacitance across the crystal terminals external to the crystal. This total capacitance external to the crystal is called the load capacitance.

As shown in Figure 5, the load capacitance is given by

\[
C_L = \frac{C_{X1} C_{X2}}{C_{X1} + C_{X2}} + C_{\text{stray}}
\]

The crystal manufacturer needs to know the value of \( C_L \) in order to adjust the crystal to the specified frequency.
The adjustment involves putting the crystal in series with the specified \( C_L \), and then "trimming" the crystal to obtain resonance of the series combination of the crystal and \( C_L \) at the specified frequency. Because of the high Q of the crystal, the resonant frequency of the series combination of the crystal and \( C_L \) is the same as the antiresonant frequency of the parallel combination of the crystal and \( C_L \).

This frequency is given by

\[
 f_s = f_c \left( 1 + \frac{C_L}{2 (C_L + C_0)} \right)
\]

These frequency formulas are derived (in Appendix I) from the equivalent circuit of the crystal, using the assumptions that the Q of the crystal is extremely high, and that the circuit external to the crystal has no effect on the frequency other than to provide the load capacitance \( C_L \). The latter assumption is not precisely true, but it is close enough for present purposes.

"Series" vs. "Parallel" Crystals

There is no such thing as a "series cut" crystal as opposed to a "parallel cut" crystal. There are different cuts of crystal, having to do with the parameters of its motional arm in various frequency ranges, but there is no special cut for series or parallel operation.

An oscillator is series resonant if the oscillation frequency is \( f_s \) of the crystal. To operate the crystal at \( f_s \), the amplifier has to be noninverting. When buying a crystal for such an oscillator, one does not specify a load capacitance. Rather, one specifies the loading condition as "series."

If a "series" crystal is put into an oscillator that has an inverting amplifier, it will oscillate in parallel resonance with the load capacitance presented to the crystal by the oscillator circuit, at a frequency slightly above \( f_s \). In fact, at approximately

\[
 f_s = f_c \left( 1 + \frac{C_L}{2 (C_L + C_0)} \right)
\]

This frequency would typically be about 0.02% above \( f_s \).

Equivalent Series Resistance

The "series resistance" often listed on quartz crystal data sheets is the real part of the crystal impedance at the crystal's calibration frequency. This will be \( R_1 \) if the calibration frequency is the series resonant frequency of the crystal. If the crystal is calibrated for parallel resonance with a load capacitance \( C_L \), the equivalent series resistance will be

\[
 \text{ESR} = R_1 \left( 1 + \frac{C_0}{C_L} \right)^2
\]

The crystal manufacturer measures this resistance at the calibration frequency during the same operation in which the crystal is adjusted to the calibration frequency.

Frequency Tolerance

Frequency tolerance as discussed here is not a requirement on the crystal, but on the complete oscillator. There are two types of frequency tolerances on oscillators: frequency accuracy and frequency stability. Frequency accuracy refers to the oscillator's ability to run at an exact specified frequency. Frequency stability refers to the constancy of the oscillation frequency.

Frequency accuracy requires mainly that the oscillator circuit present to the crystal the same load capacitance that it was adjusted for. Frequency stability requires mainly that the load capacitance be constant.

In most digital applications the accuracy and stability requirements on the oscillator are so wide that it makes very little difference what load capacitance the crystal was adjusted to, or what load capacitance the circuit actually presents to the crystal. For example, if a crystal was calibrated to a load capacitance of 25 pF, and is used in a circuit whose actual load capacitance is 50 pF, the frequency error on that account would be less than 0.01%.

In a positive reactance oscillator, the crystal only needs to be in the intended response mode for the oscillator to satisfy a 0.5% or better frequency tolerance. That's because for any load capacitance the oscillation frequency is certain to be between the crystal's resonant and antiresonant frequencies.

Phase shifts that take place within the amplifier part of the oscillator will also affect frequency accuracy and
stability. These phase shifts can normally be modeled as an "output capacitance" that, in the positive reactance oscillator, parallels \( C_{X2} \). The predictability and constancy of this output capacitance over temperature and device sample will be the limiting factor in determining the tolerances that the circuit is capable of holding.

**Drive Level**

Drive level refers to the power dissipation in the crystal. There are two reasons for specifying it. One is that the parameters in the equivalent circuit are somewhat dependent on the drive level at which the crystal is calibrated. The other is that if the application circuit exceeds the test drive level by too much, the crystal may be damaged. Note that the terms "test drive level" and "rated drive level" both refer to the drive level at which the crystal is calibrated. Normally, in a microcontroller system, neither the frequency tolerances nor the power levels justify much concern for this specification. Some crystal manufacturers don’t even require it for microprocessor crystals.

In a positive reactance oscillator, if one assumes the peak voltage across the crystal to be something in the neighborhood of \( V_{CC} \), the power dissipation can be approximated as

\[
P = 2R_s \pi f (C_L + C_0) V_{CC}^2
\]

This formula is derived in Appendix I. In a 5V system, \( P \) rarely evaluates to more than a milliwatt. Crystals with a standard 1 or 2 mW drive level rating can be used in most digital systems.

**CERAMIC RESONATORS**

Ceramic resonators operate on the same basic principles as a quartz crystal. Like quartz crystals, they are piezoelectric, have a reactance versus frequency curve similar to a crystal’s, and an equivalent circuit that looks just like a crystal’s (with different parameter values, however).

The frequency tolerance of a ceramic resonator is about two orders of magnitude wider than a crystal’s, but the ceramic is somewhat cheaper than a crystal. It may be noted for comparison that quartz crystals with relaxed tolerances cost about twice as much as ceramic resonators. For purposes of clocking a microcontroller, the frequency tolerance is often relatively noncritical, and the economic consideration becomes the dominant factor.

Figure 6 shows a graph of impedance magnitude versus frequency for a 3.58MHz ceramic resonator. (Note that Figure 6 is a graph of \( Z_r \) versus frequency, whereas Figure 3 is a graph of \( X_r \) versus frequency.) A number of spurious responses are apparent in Figure 6. The manufacturers state that spurious responses are more prevalent in the lower frequency resonators (kHz range) than in the higher frequency units (MHz range). For our purposes only the MHz range ceramics need to be considered.

![Figure 7 — Ceramic Resonator: Symbol and Equivalent Circuit](image)

Figure 7 shows the symbol and equivalent circuit for the ceramic resonator, both of which are the same as for the crystal. The parameters have different values, however, as listed in Table 2.

<table>
<thead>
<tr>
<th>frequency MHz</th>
<th>( R_1 ) ohms</th>
<th>( L_1 ) mH</th>
<th>( C_1 ) pF</th>
<th>( C_0 ) pF</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.58</td>
<td>7</td>
<td>.113</td>
<td>19.6</td>
<td>140</td>
</tr>
<tr>
<td>6.0</td>
<td>8</td>
<td>.094</td>
<td>8.3</td>
<td>60</td>
</tr>
<tr>
<td>8.0</td>
<td>7</td>
<td>.092</td>
<td>4.6</td>
<td>40</td>
</tr>
<tr>
<td>11.0</td>
<td>10</td>
<td>.057</td>
<td>3.9</td>
<td>30</td>
</tr>
</tbody>
</table>
Note that the motional arm of the ceramic resonator tends to have less resistance than the quartz crystal and also a vastly reduced \( L_i/C_j \) ratio. This results in the motional arm having a \( Q \) (given by \( (1/R_i) \sqrt{L_i/C_j} \)) that is typically two orders of magnitude lower than that of a quartz crystal. The lower \( Q \) makes for a faster start-up of the oscillator and for a less closely controlled frequency (meaning that circuitry external to the resonator will have more influence on the frequency than with a quartz crystal).

Another major difference is that the shunt capacitance of the ceramic resonator is an order of magnitude higher than \( C_0 \) of the quartz crystal and more dependent on the frequency of the resonator.

The implications of these differences are not all obvious, but some will be indicated in the section on Oscillator Calculations.

**Specifications for Ceramic Resonators**

Ceramic resonators are easier to specify than quartz crystals. All the vendor wants to know is the desired frequency and the chip you want it to work with. They'll supply the resonators, a circuit diagram showing the positions and values of other external components that may be required and a guarantee that the circuit will work properly at the specified frequency.

**Oscillator Design Considerations**

Designers of microcontroller systems have a number of options to choose from for clocking the system. The main decision is whether to use the "on-chip" oscillator or an external oscillator. If the choice is to use the on-chip oscillator, what kinds of external components are needed to make it operate as advertised? If the choice is to use an external oscillator, what type of oscillator should it be?

The decisions have to be based on both economic and technical requirements. In this section we'll discuss some of the factors that should be considered.

**On-Chip Oscillators**

In most cases, the on-chip amplifier with the appropriate external components provides the most economical solution to the clocking problem. Exceptions may arise in severe environments when frequency tolerances are tighter than about 0.01%.

The external components that need to be added are a positive reactance (normally a crystal or ceramic resonator) and the two capacitors \( C_{X1} \) and \( C_{X2} \), as shown in Figure 8.

**Crystal Specifications**

Specifications for an appropriate crystal are not very critical, unless the frequency is. Any fundamental-mode crystal of medium or better quality can be used.

We are often asked what maximum crystal resistance should be specified. The best answer to this question is the lower the better, but use what's available. The crystal resistance will have some effect on start-up time and steady-state amplitude, but not so much that it can't be compensated for by appropriate selection of the capacitances \( C_{X1} \) and \( C_{X2} \).

Similar questions are asked about specifications of load capacitance and shunt capacitance. The best advice we can give is to understand what these parameters mean and how they affect the operation of the circuit (that being the purpose of this Application Note), and then decide for yourself if such specifications are meaningful in your application or not. Normally, they're not, unless your frequency tolerances are tighter than about 0.1%.

Part of the problem is that crystal manufacturers are accustomed to talking "ppm" tolerances with radio engineers and simply won't take your order until you've filled out their list of specifications. It will help if you define your actual frequency tolerance requirements, both for yourself and to the crystal manufacturer. Don't pay for 0.003% crystals if your actual frequency tolerance is 1%.

**Oscillation Frequency**

The oscillation frequency is determined 99.5% by the crystal and up to about 0.5% by the circuit external to the crystal. The on-chip amplifier has little effect on the frequency, which is as it should be, since the amplifier parameters are temperature and process dependent.

The influence of the on-chip amplifier on the frequency is by means of its input and output (pin-to-ground) capacitances, which parallel \( C_{X1} \) and \( C_{X2} \), and the XTAL1-to-XTAL2 (pin-to-pin) capacitance, which parallels the crystal. The input and pin-to-pin capacitances are about 7 pF each. Internal phase deviations from the nominal 180° can be modeled as an output capacitance of 25 to 30 pF. These deviations from the ideal have less effect.
in the positive reactance oscillator (with the inverting amplifier) than in a comparable series resonant oscillator (with the noninverting amplifier) for two reasons: first, the effect of the output capacitance is lessened, if not swamped, by the off-chip capacitor; secondly, the positive reactance oscillator is less sensitive, frequency-wise, to such phase errors.

**Selection of Cₓ₁ and Cₓ₂**

Optimal values for the capacitors Cₓ₁ and Cₓ₂ depend on whether a quartz crystal or ceramic resonator is being used, and also on application-specific requirements on start-up time and frequency tolerance.

Start-up time is sometimes more critical in microcontroller systems than frequency stability, because of various reset and initialization requirements.

Less commonly, accuracy of the oscillator frequency is also critical, for example, when the oscillator is being used as a time base. As a general rule, fast start-up and stable frequency tend to pull the oscillator design in opposite directions.

Considerations of both start-up time and frequency stability over temperature suggest that Cₓ₁ and Cₓ₂ should be about equal and at least 20 pF. (But they don't have to be either.) Increasing the value of these capacitances above some 40 or 50 pF improves frequency stability. It also tends to increase the start-up time. There is a maximum value (several hundred pF, depending on the value of R₁ of the quartz or ceramic resonator) above which the oscillator won't start up at all.

If the on-chip amplifier is a simple inverter, such as in the 8051, the user can select values for Cₓ₁ and Cₓ₂ between some 20 and 100 pF, depending on whether start-up time or frequency stability is the more critical parameter in a specific application. If the on-chip amplifier is a Schmitt Trigger, such as in the 8048, smaller values of Cₓ₁ must be used (5 to 30 pF), in order to prevent the oscillator from running in a relaxation mode.

Later sections in this Application Note will discuss the effects of varying Cₓ₁ and Cₓ₂ (as well as other parameters), and will have more to say on their selection.

**Placement of Components**

Noise glitches arriving at the XTAL1 or XTAL2 pins at the wrong time can cause a miscount in the internal clock-generating circuitry. These kinds of glitches can be produced through capacitive coupling between the oscillator components and PCB traces carrying digital signals with fast rise and fall times. For this reason, the oscillator components should be mounted close to the chip and have short, direct traces to the XTAL1, XTAL2, and VSS pins.

**Clocking Other Chips**

There are times when it would be desirable to use the on-chip oscillator to clock other chips in the system. This can be done if an appropriate buffer is used. A TTL buffer puts too much load on the on-chip amplifier for reliable start-up. A CMOS buffer (such as the 74HC04) can be used, if it’s fast enough and if its VIH and VIL specs are compatible with the available signal amplitudes. Circuits such as shown in Figure 9 might also be considered for these types of applications.

Clock-related signals are available at the TO pin in the MCS-48 products, at ALE in the MCS-48 and MCS-51 lines, and the iACX-96 controllers provide a CLKOUT signal.

**External Oscillators**

When technical requirements dictate the use of an external oscillator, the external drive requirements for the microcontroller, as published in the data sheet, must be carefully noted. The logic levels are not in general TTL-compatible. And each controller has its idiosyncracies in this regard. The 8048, for example, requires that both XTAL1 and XTAL2 be driven. The 8051 can be driven that way, but the data sheet suggest the simpler method of grounding XTAL1 and driving XTAL2. For this method, the driving source must be capable of sinking some current when XTAL2 is being driven low.

For the external oscillator itself, there are basically two choices: ready-made and home-grown.
TTL Crystal Clock Oscillator

The HS-100, HS-200, & HS-500 all-metal package series of oscillators are TTL compatible & fit a DIP layout. Standard electrical specifications are shown below. Variations are available for special applications.

**Frequency Range:**
- HS-100 — 3.5 MHz to 30 MHz
- HS-200 — 225 KHz to 3.5 MHz
- HS-500 — 25 MHz to 60 MHz

**Frequency Tolerance:** ± .01% Overall 0-70° C

**Hermetically Sealed Package**
Mass spectrometer leak rate max.
1 x 10⁻⁸ atmos. cc/sec. of helium

<table>
<thead>
<tr>
<th>INPUT</th>
<th>HS-100</th>
<th>HS-200</th>
<th>HS-500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply Voltage (VCC)</td>
<td>5V ± 10%</td>
<td>5V ± 10%</td>
<td>5V ± 10%</td>
</tr>
<tr>
<td>Supply Current (Icc) max.</td>
<td>30mA</td>
<td>40mA</td>
<td>85mA</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>HS-100</td>
<td>HS-200</td>
<td>HS-500</td>
</tr>
<tr>
<td>VCC (Logic “1”)</td>
<td>+2.4V min.</td>
<td>+2.7V min</td>
<td>+2.4V min</td>
</tr>
<tr>
<td>VOC (Logic “0”)</td>
<td>+0.4V max.</td>
<td>+0.5V max</td>
<td>+0.4V max</td>
</tr>
<tr>
<td>Symmetry</td>
<td>60.40%</td>
<td>60.40%</td>
<td>55.45%</td>
</tr>
<tr>
<td>TR, TF (Rise &amp; Fall Time)</td>
<td>&lt; 10ns</td>
<td>&lt; 5ns</td>
<td>&lt; 15ns</td>
</tr>
<tr>
<td>Output Short</td>
<td>18mA min.</td>
<td>40mA min.</td>
<td>18mA min</td>
</tr>
<tr>
<td>Circuit Current</td>
<td>1 to 10 TTL Loads</td>
<td>1 to 10 TTL Loads</td>
<td>1 to 10 TTL Loads</td>
</tr>
<tr>
<td>Output Load</td>
<td>1 to 10 TTL Loads</td>
<td>1 to 10 TTL Loads</td>
<td>1 to 10 TTL Loads</td>
</tr>
</tbody>
</table>

** CONDITIONS **
- I0 source = 400μA max
- I0 sink = 20 mA max
- 1.6mA per load
- I0 source = 1.0mA max
- VI0 = 1.4V
- 0.2mA per load
- I0 sink = 16.0mA max
- VI0 = 4V to 2.4V

Figure 10 — Pre-packaged Oscillator Data*

Prepackaged oscillators are available from most crystal manufacturers, and have the advantage that the system designer can treat the oscillator as a black box whose performance is guaranteed by people who carry many years of experience in designing and building oscillators. Figure 10 shows a typical data sheet for some prepackaged oscillators. Oscillators are also available with complementary outputs.

If the oscillator is to drive the microcontroller directly, one will want to make a careful comparison between the external drive requirements in the microcontroller data sheet and the oscillator's output logic levels and test conditions.

If oscillator stability is less critical than cost, the user may prefer to go with an in-house design. Not without some precautions, however.

It’s easy to design oscillators that work. Almost all of them do work, even if the designer isn’t too clear on why. The key point here is that almost all of them work. The problems begin when the system goes into production, and marginal units commence malfunctioning in the field. Most digital designers, after all, are not very adept at designing oscillators for production.

Oscillator design is somewhat of a black art, with the quality of the finished product being very dependent on the designer’s experience and intuition. For that reason the most important consideration in any design is to have an adequate preproduction test program. Preproduction tests are discussed later in this Application Note. Here we will discuss some of the design options and take a look at some commonly used configurations.

*Reprinted with the permission of © Midland-Ross Corporation 1982
Gate Oscillators versus Discrete Devices

Digital systems designers are understandably reluctant to get involved with discrete devices and their peculiarities (biasing techniques, etc.). Besides, the component count for these circuits tends to be quite a bit higher than what a digital designer is used to seeing for that amount of functionality. Nevertheless, if there are unusual requirements on the accuracy and stability of the clock frequency, it should be noted that discrete device oscillators can be tailored to suit the exact needs of the application and perfected to a level that would be difficult for a gate oscillator to approach.

In most cases, when an external oscillator is needed, the designer tends to rely on some form of a gate oscillator. A TTL inverter with a resistor connecting the output to the input makes a suitable inverting amplifier. The resistor holds the inverter in the transition region between logical high and low, so that at least for start-up purposes the inverter is a linear amplifier.

The feedback resistance has to be quite low, however, since it must conduct current sourced by the input pin without allowing the DC input voltage to get too far above the DC output voltage. For biasing purposes, the feedback resistance should not exceed a few k-ohms.

But shunting the crystal with such a low resistance does not encourage start-up.

Consequently, the configuration in Figure 11A might be suggested. By breaking $R_f$ into two parts and AC-grounding the midpoint, one achieves the DC feedback required to hold the inverter in its active region, but without the negative signal feedback that is in effect telling the circuit not to oscillate. However, this biasing scheme will increase the start-up time, and relaxation-type oscillations are also possible.

A CMOS inverter, such as the 74HC04, might work better in this application, since a larger $R_f$ can be used to hold the inverter in its linear region.

Logic gates tend to have a fairly low output resistance, which destabilizes the oscillator. For that reason a resistor $R_x$ is often added to the feedback network, as shown in Figure 11A and B. At higher frequencies a 20 or 30 pF capacitor is sometimes used in the $R_x$ position, to compensate for some of the internal propagation delay.

Reference 1 contains an excellent discussion of gate oscillators, and a number of design examples.

**Fundamental versus Overtone Operation**

It's easier to design an oscillator circuit to operate in the resonator's fundamental response mode than to design one for overtone operation. A quartz crystal whose fundamental response mode covers the desired frequency can be obtained up to some 30 MHz. For frequencies above that, the crystal might be used in an overtone mode.

Several problems arise in the design of an overtone oscillator. One is to stop the circuit from oscillating in the fundamental mode, which is what it would really rather do, for a number of reasons, involving both the amplifying device and the crystal. An additional problem with overtone operation is an increased tendency to spurious oscillations. That is because the $R_f$ of various spurious modes is likely to be about the same as $R_f$ of the intended overtone response. It may be necessary, as suggested in reference 1, to specify a "spurious-to-main-response" resistance ratio to avoid the possibility of trouble.

Overtone oscillators are not to be taken lightly. One would be well advised to consult with an engineer who is knowledgeable in the subject during the design phase of such a circuit.

**Series versus Parallel Operation**

Series resonant oscillators use noninverting amplifiers. To make a noninverting amplifier out of logic gates requires that two inverters be used, as shown in Figure 12.

This type of circuit tends to be inaccurate and unstable.
in frequency over variations in temperature and VCC. It has a tendency to oscillate at overtones, and to oscillate through $C_0$ of the crystal or some stray capacitance rather than as controlled by the mechanical resonance of the crystal.

The demon in series resonant oscillators is the phase shift in the amplifier. The series resonant oscillator wants more than just a "noninverting" amplifier—it wants a zero phase-shift amplifier. Multistage noninverting amplifiers tend to have a considerably lagging phase shift, such that the crystal reactance must be capacitive in order to bring the total phase shift around the feedback loop back up to 0. In this mode, a "12 MHz" crystal may be running at 8 or 9 MHz. One can put a capacitor in series with the crystal to relieve the crystal of having to produce all of the required phase shift, and bring the oscillation frequency closer to $f_s$. However, to further complicate the situation, the amplifier's phase shift is strongly dependent on frequency, temperature, VCC, and device sample.

Positive reactance oscillators ("parallel resonant") use inverting amplifiers. A single logic inverter can be used for the amplifier, as in Figure 11. The amplifier's phase shift is less critical, compared to the series resonant circuit, and since only one inverter is involved there's less phase error anyway. The oscillation frequency is effectively bounded by the resonant and antiresonant frequencies of the crystal itself. In addition, the feedback network includes capacitors that parallel the input and output terminals of the amplifier, thus reducing the effect of unpredictable capacitances at these points.

**MORE ABOUT USING THE "ON-CHIP" OSCILLATORS**

In this section we will describe the on-chip inverters on selected microcontrollers in some detail, and discuss criteria for selecting components to work with them. Future data sheets will supplement this discussion with updates and information pertinent to the use of each chip's oscillator circuitry.

**Oscillator Calculations**

Oscillator design, though aided by theory, is still largely an empirical exercise. The circuit is inherently nonlinear, and the normal analysis parameters vary with instantaneous voltage. In addition, when dealing with the on-chip circuitry, we have FETs being used as resistors, resistors being used as interconnects, distributed delays, input protection devices, parasitic junctions, and processing variations.

Consequently, oscillator calculations are never very precise. They can be useful, however, if they will at least indicate the effects of variations in the circuit parameters on start-up time, oscillation frequency, and steady-state amplitude. Start-up time, for example, can be taken as an indication of start-up reliability. If preproduction tests indicate a possible start-up problem, a relatively inexperienced designer can at least be made aware of what
parameter may be causing the marginality, and what direction to go in to fix it.

The analysis used here is mathematically straightforward but algebraically intractable. That means it's relatively easy to understand and program into a computer, but it will not yield a neat formula that gives, say, steady-state amplitude as a function of this or that list of parameters. A listing of a BASIC program that implements the analysis will be found in Appendix II.

When the circuit is first powered up, and before the oscillations have commenced (and if the oscillations fail to commence), the oscillator can be treated as a small signal linear amplifier with feedback. In that case, standard small-signal analysis techniques can be used to determine start-up characteristics. The circuit model used in this analysis is shown in Figure 13.

The circuit approximates that there are no high-frequency effects within the amplifier itself, such that its high-frequency behavior is dominated by the load impedance $Z_L$. This is a reasonable approximation for single-stage amplifiers of the type used in 8051-type devices. Then the gain of the amplifier as a function of frequency is

$$A = \frac{A_1 Z_L}{Z_L + R_0}$$

The gain of the feedback network is

$$\beta = \frac{Z_i}{Z_i + Z_f}$$

And the loop gain is

$$\beta A = \frac{1}{Z_i + Z_f} \cdot \frac{A_1 Z_L}{Z_L + R_0}$$

The impedances $Z_L$, $Z_i$, and $Z_f$ are defined in Figure 13B.

Figure 14 shows the way the loop gain thus calculated (using typical 8051-type parameters and a 4.608 MHz crystal) varies with frequency. The frequency of interest is the one for which the phase of the loop gain is zero. The accepted criterion for start-up is that the magnitude of the loop gain must exceed unity at this frequency. This is the frequency at which the circuit is in resonance. It corresponds very closely with the antiresonant frequency of the motional arm of the crystal in parallel with $C_L$.

Figure 15 shows the way the loop gain varies with frequency when the parameters of a 3.58 MHz ceramic resonator are used in place of a crystal (the amplifier parameters being typical 8051, as in Figure 14). Note the different frequency scales.
Start-Up Characteristics

It is common, in studies of feedback systems, to examine the behavior of the closed loop gain as a function of complex frequency $s = \sigma + j\omega$; specifically, to determine the location of its poles in the complex plane. A pole is a point on the complex plane where the gain function goes to infinity. Knowledge of its location can be used to predict the response of the system to an input disturbance.

The way that the response function depends on the location of the poles is shown in Figure 16. Poles in the left half plane cause the response function to take the form of a damped sinusoid. Poles in the right half plane cause the response function to take the form of an exponentially growing sinusoid. In general,

$$v(t) \sim e^{a t}\sin(\omega t + \Theta)$$

where $a$ is the real part of the pole frequency. Thus if the pole is in the right half plane, $a$ is positive and the sinusoid grows. If the pole is in the left half plane, $a$ is negative and the sinusoid is damped.

The same type of analysis can usefully be applied to oscillators. In this case, however, rather than trying to ensure that the poles are in the left half plane, we would seek to ensure that they're in the right half plane. An exponentially growing sinusoid is exactly what is wanted from an oscillator that has just been powered up.

The gain function of interest in oscillators is $1/(1 - \beta A)$. Its poles are at the complex frequencies where $\beta A = 1/|\theta|^2$, because that value of $\beta A$ causes the gain function to go to infinity. The oscillator will start up if the real part of the pole frequency is positive. More importantly, the rate at which it starts up is indicated by how much greater than 0 the real part of the pole frequency is.

![Figure 16](image)

**Figure 16 — Do You Know Where Your Poles are Tonight?**

![Figure 17](image)

**Figure 17 — Oscillator Start-Up (4.608 MHz Crystal from Standard Crystal Corp.)**
The circuit in Figure 13B can be used to find the pole frequencies of the oscillator gain function. All that needs to be done is evaluate the impedances at complex frequencies \( \sigma + j\omega \) rather than just at \( \omega \), and find the value of \( \sigma + j\omega \) for which \( \beta A = 1 / \sigma \). The larger that value of \( \sigma \) is, the faster the oscillator will start up.

Of course, other things besides pole frequencies, things like the VCC rise time, are at work in determining the start-up time. But to the extent that the pole frequencies do affect start-up time, we can obtain results like those in Figures 17 and 18.

To obtain these figures, the pole frequencies were computed for various values of capacitance \( C_x \) from XTAL1 and XTAL2 to ground (thus \( C_{x1} = C_{x2} = C_x \)). Then a “time constant” for start-up was calculated as

\[ T_s = \frac{1}{\sigma} \]

where \( \sigma \) is the real part of the pole frequency (rad/sec), and this time constant is plotted versus \( C_x \).

A short time constant means faster start-up. A long time constant means slow start-up. Observations of actual start-ups are shown in the figures. Figure 17 is for a typical 8051 with a 4.608 MHz crystal supplied by Standard Crystal Corp., and Figure 18 is for a typical 8051 with a 3.58 MHz ceramic resonator supplied by NTK Technical Ceramics, Ltd.

It can be seen in Figure 17 that, for this crystal, values of \( C_x \) between 30 and 50 pF minimize start-up time, but that the exact value in this range is not particularly important, even if the start-up time itself is critical.

As previously mentioned, start-up time can be taken as an indication of start-up reliability. Start-up problems are normally associated with \( C_{x1} \) and \( C_{x2} \) being too small or too large for a given resonator. If the parameters of the resonator are known, curves such as in Figure 17 or 18 can be generated to define acceptable ranges of values for these capacitors.

As the oscillations grow in amplitude, they reach a level at which they undergo severe clipping within the amplifier, in effect reducing the amplifier gain. As the amplifier gain decreases, the poles move towards the \( j\omega \) axis. In steady-state, the poles are on the \( j\omega \) axis and the amplitude of the oscillations is constant.

**Steady-State Characteristics**

Steady-state analysis is greatly complicated by the fact that we are dealing with large signals and nonlinear circuit response. The circuit parameters vary with instantaneous voltage, and a number of clamping and clipping mechanisms come into play. Analyses that take all these things into account are too complicated to be of general use, and analyses that don’t take them into account are too inaccurate to justify the effort.

There is a steady-state analysis in Appendix II that takes some of the complications into account and ignores others. Figure 19 shows the way the steady-state amplitudes thus calculated (using typical 8051 parameters and a 4.608 MHz crystal) vary with equal bulk capacitance placed from XTAL1 and XTAL2 to ground. Experimental results are shown for comparison.

The waveform at XTAL1 is a fairly clean sinusoid. Its negative peak is normally somewhat below zero, at a level which is determined mainly by the input protection circuitry at XTAL1.
The effect of the input protection circuitry on the oscillator is that if the XTAL1 signal goes negative, its negative peak is clamped to \(-V_{DS}\) of the protection FET in the ROM parts, and to about \(-.5V\) in the EPROM parts. These negative voltages on XTAL1 are in this application self-limiting and nondestructive.

The clamping action does, however, raise the DC level at XTAL1, which in turn tends to reduce the positive peak at XTAL2. The waveform at XTAL2 resembles a sinusoid riding on a DC level, and whose negative peaks are clipped off at zero.

Since it's normally the XTAL2 signal that drives the internal clocking circuitry, the question naturally arises as to how large this signal must be to reliably do its job. In fact, the XTAL2 signal doesn't have to meet the same VIH and VIL specifications that an external driver would have to. That's because as long as the oscillator is working, the on-chip amplifier is driving itself through its own 0-to-1 transition region, which is very nearly the same as the 0-to-1 transition region in the internal buffer that follows the oscillator. If some processing variations move the transition level higher or lower, the on-chip amplifier tends to compensate for it by the fact that its own transition level is correspondingly higher or lower. (In the 8096, it's the XTAL1 signal that drives the internal clocking circuitry, but the same concept applies.)

The main concern about the XTAL2 signal amplitude is as an indication of the general health of the oscillator. An amplitude of less than about 2.5V peak-to-peak indicates that start-up problems could develop in some units (with low gain) with some crystals (with high \(R_1\)). The remedy is to either adjust the values of \(C_{X1}\) and/or \(C_{X2}\) or use a crystal with a lower \(R_1\).

The amplitudes at XTAL1 and XTAL2 can be adjusted by changing the ratio of the capacitors from XTAL1 and XTAL2 to ground. Increasing the XTAL2 capacitance, for example, decreases the amplitude at XTAL2 and increases the amplitude at XTAL1 by about the same amount. Decreasing both caps increases both amplitudes.

**Pin Capacitance**

Internal pin-to-ground and pin-to-pin capacitances at XTAL1 and XTAL2 will have some effect on the oscillator. These capacitances are normally taken to be in the range of 5 to 10 pF, but they are extremely difficult to evaluate. Any measurement of one such capacitance will necessarily include effects from the others. One advantage of the positive reactance oscillator is that the pin-to-ground capacitances are paralleled by external bulk capacitors, so a precise determination of their value is unnecessary. We would suggest that there is little justification for more precision than to assign them a value of 7 pF (XTAL1-to-ground and XTAL1-to-XTAL2). This value is probably not in error by more than 3 or 4 pF.

The XTAL2-to-ground capacitance is not entirely "pin capacitance," but more like an "equivalent output capacitance" of some 25 to 30 pF, having to include the effect of internal phase delays. This value will vary to some extent with temperature, processing, and frequency.
MC5®-51 Oscillator

The on-chip amplifier on the HMOS MCS-51 family is shown in Figure 20. The drain load and feedback "resistors" are seen to be field-effect transistors. The drain load FET, $R_D$, is typically equivalent to about 1 k to 3k-ohms. As an amplifier, the low frequency voltage gain is normally between $-10$ and $-20$, and the output resistance is effectively $R_D'$. The 8051 oscillator is normally used with equal bulk capacitors placed externally from XTAL1 to ground and from XTAL2 to ground. To determine a reasonable value of capacitance to use in these positions, given a crystal or ceramic resonator of known parameters, one can use the BASIC analysis in Appendix II to generate curves such as in Figures 17' and 18. This procedure will define a range of values that will minimize start-up time. We don't suggest that smaller values be used than those which minimize start-up time. Larger values than those can be used in applications where increased frequency stability is desired, at some sacrifice in start-up time.

The 8051 oscillator is normally used with equal bulk capacitors placed externally from XTAL1 to ground and from XTAL2 to ground. To determine a reasonable value of capacitance to use in these positions, given a crystal or ceramic resonator of known parameters, one can use the BASIC analysis in Appendix II to generate curves such as in Figures 17 and 18. This procedure will define a range of values that will minimize start-up time. We don't suggest that smaller values be used than those which minimize start-up time. Larger values than those can be used in applications where increased frequency stability is desired, at some sacrifice in start-up time.

Figure 20 — MCS®-51 Oscillator Amplifier

Standard Crystal Corp. (reference 8) studied the use of their crystals with the MCS-51 family using skew samples supplied by Intel. They suggest putting 30 pF capacitors from XTAL1 and XTAL2 to ground, if the crystal is specified as described in reference 8. They noted that in that configuration and with crystals thus specified, the frequency accuracy was $\pm 0.01\%$ and the frequency stability was $\pm 0.005\%$, and that a frequency accuracy of $\pm 0.005\%$ could be obtained by substituting a 25 pF fixed cap in parallel with a 5-20 pF trimmer for one of the 30 pF caps.

MC5-51 skew samples have also been supplied to a number of ceramic resonator manufacturers for characterization with their products. These companies should be contacted for application information on their products. In general, however, ceramics tend to want somewhat larger values for $C_{X1}$ and $C_{X2}$ than quartz crystals do. As shown in Figure 18, they start up a lot faster that way.

In some applications the actual frequency tolerance required is only 1% or so, the user being concerned mainly that the circuit will oscillate. In that case, $C_{X1}$ and $C_{X2}$ can be selected rather freely in the range of 20 to 80 pF.

As you can see, 'best' values for these components and their tolerances are strongly dependent on the application and its requirements. In any case, their suitability should be verified by environmental testing before the design is submitted to production.

MCS®-48 Oscillator

The NMOS and HMOS MCS-48 oscillator is shown in Figure 21. It differs from the 8051 in that its inverting amplifier is a Schmitt Trigger. This configuration was chosen to prevent crosstalk from the TO pin, which is adjacent to the XTAL1 pin.

All Schmitt Trigger circuits exhibit a hysteresis effect, as shown in Figure 22. The hysteresis is what makes it less sensitive to noise. The same hysteresis allows any Schmitt Trigger to be used as a relaxation oscillator. All you have to do is connect a resistor from output to input, and a capacitor from input to ground, and the circuit oscillates in a relaxation mode as follows.

If the Schmitt Trigger output is at a logic high, the capacitor commences charging through the feedback resistor. When the capacitor voltage reaches the upper trigger point (UTP), the Schmitt Trigger output switches to a logic low and the capacitor commences discharging.
through the same resistor. When the capacitor voltage reaches the lower trigger point (LTP), the Schmitt Trigger output switches to a logic high again, and the sequence repeats. The oscillation frequency is determined by the RC time constant and the hysteresis voltage, UTP-LTP.

The 8048 can oscillate in this mode. It has an internal feedback resistor. All that's needed is an external capacitor from XTAL1 to ground. In fact, if a smaller external feedback resistor is added, an 8048 system could be designed to run in this mode. Do it at your own risk! This mode of operation is not tested, specified, documented, or encouraged in any way by Intel for the 8048. Future stepping of the device might have a different type of inverting amplifier (one more like the 8051). The CHMOS members of the MCS-48 family do not use a Schmitt Trigger as the inverting amplifier.

Relaxation oscillations in the 8048 must be avoided, and this is the major objective in selecting the off-chip components needed to complete the oscillator circuit.

When an 8048 is powered up, if VCC has a short rise time, the relaxation mode starts first. The frequency is normally about 50kHz. The resonator mode builds more slowly, but it eventually takes over and dominates the operation of the circuit. This is shown in Figure 23A.

Due to processing variations, some units seem to have a harder time coming out of the relaxation mode, particularly at low temperatures. In some cases the resonator oscillations may fail entirely, and leave the device in the relaxation mode. Most units will stick in the relaxation mode at any temperature if \( C_{X1} \) is larger than about 50 pF. Therefore, \( C_{X1} \) should be chosen with some care, particularly if the system must operate at lower temperatures.

One method that has proven effective in all units to \(-40 \text{ C}\) is to put 5 pF from XTAL1 to ground and 20 pF from XTAL2 to ground. Unfortunately, while this method does discourage the relaxation mode, it is not an optimal choice for the resonator mode. For one thing, it does not swamp the pin capacitance. Also, it makes for a rather high signal level at XTAL1 (8 or 9 volts peak-to-peak).

The question arises as to whether that level of signal at XTAL1 might damage the chip. Not to worry. The negative peaks are self-limiting and nondestructive. The positive peaks could conceivably damage the oxide, but in fact, NMOS chips (eg, 8048) and HMOS chips (eg, 8048H) are tested to a much higher voltage than that. The technology trend, of course, is to thinner oxides, as the devices shrink in size. For an extra margin of safety, the HMOS II chips (eg, 8048AH) have an internal diode clamp at XTAL1 to VCC.

In reality, \( C_{X1} \) doesn't have to be quite so small to avoid relaxation oscillations, if the minimum operating temperature is not \(-40 \text{ C}\). For less severe temperature requirements, values of capacitance selected in much the same way as for an 8051 can be used. The circuit should be tested, however, at the system's lowest temperature limit.

Additional security against relaxation oscillations can be obtained by putting a 1M-ohm (or larger) resistor from XTAL1 to VCC. Pulling up the XTAL1 pin this way seems to discourage relaxation oscillations as effectively as any other method (Figure 23B).

Another thing that discourages relaxation oscillations is low VCC. The resonator mode, on the other hand, is much less sensitive to VCC. Thus if VCC comes up relatively slowly (several milliseconds rise time), the resonator mode is normally up and running before the relaxation mode starts (in fact, before VCC has even reached operating specs). This is shown in Figure 23C.

A secondary effect of the hysteresis is a shift in the oscillation frequency. At low frequencies, the output signal from an inverter without hysteresis leads (or lags) the input by 180 degrees. The hysteresis in a Schmitt Trigger, however, causes the output to lead the input by less than 180 degrees (or lag by more than 180 degrees), by an amount that depends on the signal amplitude, as shown in Figure 24. At higher frequencies, there are additional phase shifts due to the various reactances in the circuit, but the phase shift due to the hysteresis is still present. Since the total phase shift in the oscillator's loop gain is necessarily 0 or 360 degrees, it is apparent that as the oscillations build up, the frequency has to change to allow the reactances to compensate for the hysteresis. In normal operation, this additional phase shift due to hysteresis does not exceed a few degrees, and the resulting frequency shift is negligible.

Kyocera, a ceramic resonator manufacturer, studied the
use of some of their resonators (at 6.0MHz, 8.0MHz, and 11.0MHz) with the 8049H. Their conclusion as to the value of capacitance to use at XTAL1 and XTAL2 was that 33 pF is appropriate at all three frequencies. One should probably follow the manufacturer's recommendations in this matter, since they will guarantee operation.

Whether one should accept these recommendations and guarantees without further testing is, however, another

Figure 23 — Relaxation Oscillations in the 8048
A) Inverter without hysteresis: output leads input by 180°.

B) Inverter with hysteresis: output leads input by less than 180°.

Figure 24 — Amplitude — Dependent Phase Shift in Schmitt Trigger

A) software for oscillator test

B) oscillator test circuit (shown for 8051 test)

Preproduction Tests

An oscillator design should never be considered ready for production until it has proven its ability to function acceptably well under worst-case environmental conditions and with parameters at their worst-case tolerance limits. Unexpected temperature effects in parts that may already be near their tolerance limits can prevent start-up of an oscillator that works perfectly well on the bench. For example, designers often overlook temperature effects in ceramic capacitors. (Some ceramics are down to 50% of their room-temperature values at –20° C and +60° C.) The problem here isn’t just one of
frequency stability, but also involves start-up time and steady-state amplitude. There may also be temperature effects in the resonator and amplifier.

It will be helpful to build a test jig that will allow the oscillator circuit to be tested independently of the rest of the system. Both start-up and steady-state characteristics should be tested. Figure 25 shows the circuit that was used to obtain the oscillator start-up photographs in this Application Note. This circuit or a modified version of it would make a convenient test vehicle. The oscillator and its relevant components can be physically separated from the control circuitry, and placed in a temperature chamber.

Start-up should be observed under a variety of conditions, including low VCC and using slow and fast VCC rise times. The oscillator should not be reluctant to start up even when VCC is below its spec value for the rest of the chip. (The rest of the chip may not function, but the oscillator should work.) It should also be verified that start-up occurs when the resonator has more than its upper tolerance limit of series resistance. (Put some resistance in series with the resonator for this test.) The bulk capacitors from XTAL1 and XTAL2 to ground should also be varied to their tolerance limits.

The same circuit, with appropriate changes in the software to lengthen the "on" time, can be used to test the steady-state characteristics of the oscillator, specifically the frequency, frequency stability, and amplitudes at XTAL1 and XTAL2.

As previously noted, the voltage swings at these pins are not critical, but they should be checked at the system's temperature limits to ensure that they are in good health. Observing these signals necessarily changes them somewhat. Observing the signal at XTAL2 requires that the capacitor at that pin be reduced to account for the oscilloscope probe capacitance. Observing the signal at XTAL1 requires the same consideration, plus a blocking capacitor (switch the oscilloscope input to AC), so as to not disturb the DC level at that pin. Alternatively, a MOSFET buffer such as the one shown in Figure 26 can be used. It should be verified by direct measurement that the ground clip on the scope probe is ohmically connected to the scope chassis (probes are incredibly fragile in this respect), and the observations should be made with the ground clip on the VSS pin, or very close to it. If the probe shield isn't operational and in use, the observations are worthless.

Frequency checks should be made with only the oscillator circuitry connected to XTAL1 and XTAL2. The ALE frequency can be counted, and the oscillator frequency derived from that. In systems where the frequency tolerance is only "nominal," the frequency should still be checked to ascertain that the oscillator isn't running in a spurious resonance or relaxation mode. Switching VCC off and on again repeatedly will help reveal a tendency to go into unwanted modes of oscillation.

The operation of the oscillator should then be verified under actual system running conditions. By this stage one will be able to have some confidence that the basic selection of components for the oscillator itself is suitable, so if the oscillator appears to malfunction in the system the fault is not in the selection of these components.

Troubleshooting Oscillator Problems

The first thing to consider in case of difficulty is that between the test jig and the actual application there may be significant differences in stray capacitances, particularly if the actual application is on a multi-layer board. Noise glitches, that aren't present in the test jig but are in the application board, are another possibility. Capacitive coupling between the oscillator circuitry and other signals has already been mentioned as a source of miscounts in the internal clocking circuitry. Inductive coupling is also possible, if there are strong currents nearby. These problems are a function of the PCB layout.

Surrounding the oscillator components with "quiet" traces (VCC and ground, for example) will alleviate capacitive coupling to signals that have fast transition times. To minimize inductive coupling, the PCB layout should minimize the areas of the loops formed by the oscillator components. These are the loops that should be checked:

- XTAL1 through the resonator to XTAL2;
- XTAL1 through CX1 to the VSS pin;
- XTAL2 through CX2 to the VSS pin.

Figure 26 — MOSFET Buffer for Observing Oscillator Signals
It is not unusual to find that the grounded ends of $C_{x1}$ and $C_{x2}$ eventually connect up to the VSS pin only after looping around the farthest ends of the board. Not good. Finally, it should not be overlooked that software problems sometimes imitate the symptoms of a slow-starting oscillator or incorrect frequency. Never underestimate the perversity of a software problem.

REFERENCES

APPENDIX I .................. 22
APPENDIX II .................. 24
APPENDIX I
QUARTZ AND CERAMIC RESONATOR FORMULAS

Based on the equivalent circuit of the crystal, the impedance of the crystal is

\[
Z_{\text{XTAL}} = \frac{(R_1 + j\omega L_1 + 1/j\omega C_1)(1/j\omega C_0)}{R_1 + j\omega L_1 + 1/j\omega C_1 + 1/j\omega C_0}
\]

After some algebraic manipulation, this calculation can be written in the form

\[
Z_{\text{XTAL}} = \frac{1}{j\omega(C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}
\]

where \(C_T\) is the capacitance of \(C_1\) in series with \(C_0\):

\[
C_T = \frac{C_1 C_0}{C_1 + C_0}
\]

The impedance of the crystal in parallel with an external load capacitance \(C_L\) is the same expression, but with \(C_0 + C_L\) substituted for \(C_0\):

\[
Z_{\text{XTAL} + \text{CL}} = Z_{\text{XTAL}} + \frac{1}{j\omega C_L} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}
\]

where \(C_T\) is the capacitance of \(C_1\) in series with \((C_0 + C_L)\):

\[
C_T = \frac{C_1(C_0 + C_L)}{C_1 + C_0 + C_L}
\]

The impedance of the crystal in series with the load capacitance is

\[
Z_{\text{XTAL} \parallel \text{CL}} = Z_{\text{XTAL}} + \frac{1}{j\omega C_L} \cdot \frac{C_L + C_1 + C_0}{j\omega C_L (C_1 + C_0)} \cdot \frac{1 - \omega^2 L_1 C_1 + j\omega R_1 C_1}{1 - \omega^2 L_1 C_T + j\omega R_1 C_T}
\]

where \(C_T\) and \(C_T'\) are as defined above.

The phase angles of these impedances are readily obtained from the impedance expressions themselves:

\[
\Theta_{\text{XTAL}} = \arctan \frac{\omega R_1 C_T}{1 - \omega^2 L_1 C_T} - \arctan \frac{\omega R_1 C_T'}{1 - \omega^2 L_1 C_T'}\quad - \frac{\pi}{2}
\]

\[
\Theta_{\text{XTAL} \parallel \text{CL}} = \arctan \frac{\omega R_1 C_1}{1 - \omega^2 L_1 C_1} - \arctan \frac{\omega R_1 C_1'}{1 - \omega^2 L_1 C_1'}\quad - \frac{\pi}{2}
\]

The resonant (''series resonant'') frequency is the frequency at which the phase angle is zero and the impedance is low. The antiresonant (''parallel resonant'') frequency is the frequency at which the phase angle is zero and the impedance is high.

Each of the above \(\Theta\)-expressions contains two arctan functions. Setting the denominator of the argument of the first arctan function to zero gives (approximately) the ''series resonant'' frequency for that configuration. Setting the denominator of the argument of the second arctan function to zero gives (approximately) the ''parallel resonant'' frequency for that configuration.

For example, the resonant frequency of the crystal is the frequency at which

\[
1 - \omega^2 L_1 C_1 = 0
\]

Thus,

\[
\omega_s = \frac{1}{\sqrt{L_1 C_1}}
\]

or

\[
f_s = \frac{1}{2\pi \sqrt{L_1 C_1}}
\]

It will be noted that the series resonant frequency of the ''XTAL + CL'' configuration (crystal in series with \(C_L\)) is the same as the parallel resonant frequency of the ''XTAL // CL'' configuration (crystal in parallel with \(C_L\)). This is the frequency at which

\[
1 - \omega^2 L_1 C_T' = 0
\]

Thus,

\[
\omega_s = \frac{1}{\sqrt{L_1 C_T'}}
\]

or

\[
f_s = \frac{1}{2\pi \sqrt{L_1 C_T'}}
\]

This fact is used by crystal manufacturers in the process of calibrating a crystal to a specified load capacitance.

By subtracting the resonant frequency of the crystal from its antiresonant frequency, one can calculate the range of frequencies over which the crystal reactance is positive:

\[
f_s - f_a = f_a (\sqrt{1 + C_1/C_0} - 1)
\]

\[
n = \frac{C_1}{2C_0}
\]

22-46
Given typical values for $C_0$ and $C_L$, this range can hardly exceed 0.5% of $f_s$. Unless the inverting amplifier in the positive reactance oscillator is doing something very strange indeed, the oscillation frequency is bound to be accurate to that percentage whether the crystal was calibrated for series operation or to any unspecified load capacitance.

**Equivalent Series Resistance**

ESR is the real part of $Z_{XTAL}$ at the oscillation frequency. The oscillation frequency is the parallel resonant frequency of the "XTAL || CL" configuration (which is the same as the series resonant frequency of the "XTAL + CL" configuration). Substituting this frequency into the $Z_{XTAL}$ expression yields, after some algebraic manipulation,

$$\text{ESR} = \frac{R_1 \left( \frac{C_0 + C_L}{C_L} \right)^2}{1 + \omega^2 C_L^2 \left( \frac{C_0 + C_L}{C_L} \right)^2} = R_1 \left( 1 + \frac{C_0}{C_L} \right)^2$$

**Drive Level**

The power dissipated by the crystal is $I_1 R_1$, where $I_1$ is the RMS current in the motional arm of the crystal. This current is given by $V_x / |Z|$, where $V_x$ is the RMS voltage across the crystal, and $|Z|$ is the magnitude of the impedance of the motional arm. At the oscillation frequency, the motional arm is a positive (inductive) reactance in parallel resonance with $(C_0 + C_L)$. Therefore $|Z|$ is approximately equal to the magnitude of the reactance of $(C_0 + C_L)$:

$$|Z| = \frac{1}{2\pi f (C_0 + C_L)}$$

where $f$ is the oscillation frequency. Then,

$$P = I_1 R_1 = \left( \frac{V_x}{|Z|} \right)^2 R_1 = \left( \frac{2\pi f (C_0 + C_L) V_x}{|Z|} \right)^2 R_1$$

The waveform of the voltage across the crystal (XTAL1 to XTAL2) is approximately sinusoidal. If its peak value is $V_{CC}$, then $V_x$ is $V_{CC} / \sqrt{2}$. Therefore,

$$P = 2R_1 \left( \frac{2\pi f (C_0 + C_L) V_{CC}}{|Z|} \right)^2$$
APPENDIX II
OSCILLATOR ANALYSIS PROGRAM

The program is written in BASIC. BASIC is excruciatingly slow, but it has some advantages. For one thing, more people know BASIC than FORTRAN. In addition, a BASIC program is easy to develop, modify, and "fiddle around" with. Another important advantage is that a BASIC program can run on practically any small computer system.

Its slowness is a problem, however. For example, the routine which calculates the "start-up time constant" discussed in the text may take several hours to complete. A person who finds this program useful may prefer to convert it to FORTRAN, if the facilities are available.

Limitations of the Program

The program was developed with specific reference to 8051-type oscillator circuitry. That means the on-chip amplifier is a simple inverter, and not a Schmitt Trigger. The 8096, the 80C51, the 80C48 and 80C49 all have simple inverters. The 8096 oscillator is almost identical to the 8051, differing mainly in the input protection circuitry. The CHMOS amplifiers have somewhat different parameters (higher gain, for example), and different transition levels than the 8051.

The MCS-48 family is specifically included in the program only to the extent that the input-output curve used in the steady-state analysis is that of a Schmitt Trigger, if the user identifies the device under analysis as an MCS-48 device. The analysis does not include the voltage dependent phase shift of the Schmitt Trigger.

The clamping action of the input protection circuitry is important in determining the steady-state amplitudes. The steady-state routine accounts for it by setting the negative peak of the XTAL1 signal at a level which depends on the amplitude of the XTAL1 signal in accordance with experimental observations. It's an exercise in curve-fitting. A user may find a different type of curve works better. Later steppings of the chips may behave differently in this respect, having somewhat different types of input protection circuitry.

It should be noted that the analysis ignores a number of important items, such as high-frequency effects in the on-chip circuitry. These effects are difficult to predict, and are no doubt dependent on temperature, frequency, and device sample. However, they can be simulated to a reasonable degree by adding an "output capacitance" of about 20 pF to the circuit model (ie, in parallel with CX2), as described below.

Notes on Using the Program

The program asks the user to input values for various circuit parameters. First the crystal (or ceramic resonator) parameters are asked for. These are R1, L1, C1, and C0. The manufacturer can supply these values for selected samples. To obtain any kind of correlation between calculation and experiment, the values of these parameters must be known for the specific sample in the test circuit. The value that should be entered for C0 is the C0 of the crystal itself plus an estimated 7 pF to account for the XTAL1-to-XTAL2 pin capacitance, plus any other stray capacitance paralleling the crystal that the user may feel is significant enough to be included.

Then the program asks for the values of the XTAL1-to-ground and XTAL2-to-ground capacitances. For CXTAL1, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance. For CXTAL2, enter the value of the externally connected bulk capacitor plus an estimated 7 pF for pin capacitance plus about 20 pF to simulate high-frequency roll-off and phase shifts in the on-chip circuitry.

Next the program asks for values for the small-signal parameters of the on-chip amplifier. Typically, for the 8051/8751,

- **Amplifier Gain Magnitude** = 15
- **Feedback Resistance** = 2300k-ohms
- **Output Resistance** = 2k-ohms

The same values can be used for MCS-48 (NMOS and HMOs) devices, but they are difficult to verify, because the Schmitt Trigger does not lend itself to small-signal measurements.
FUNCTIONS

FUNCTION FNZM (R, X)
FNZM = SQR(R^2 + X^2)
END

FUNCTION FNZP (R, X)
FNZP = 180/PI * ATN(X/R)
END

FUNCTION FNRL (S, L)
FNRL = 2# * PI * S * L
END

FUNCTION FNXL (F, L)
FNXL = 2# * PI * F * L
END

FUNCTION FNRC (S, C, C)
FNRC = S/(2# * PI * T(S^2 + F^2) * C)
END

FUNCTION FNXR (S, C, C)
FNXR = F/(2# * PI * T(S^2 + F^2) * C)
END

FUNCTION FNRR (RA, XA, RB, XB)
FNRR = (RA * RB + XA * XB)/(RB^2 + XB^2)
END

FUNCTION FNXR (RA, XA, RB, XB)
FNXR = (XA * RB - XB * RA)/(RB^2 + XB^2)
END

BEGIN COMPUTATIONS
LET PI = 3.141592654#

END
PRINT
INPUT N
IF N=1 THEN PRINT ELSE 8600
8700 REM
8800 REM ------------------ LIST PARAMETERS ----------------------------
8900 GOSUB 17100
9000 GOTO 6800
9100 IF N=2 THEN PRINT ELSE 9400
9200 REM
9300 REM ------------------ OSCILLATION FREQUENCY ------------------------
9400 CL = CX*CY/(CX+CY) + CO
9500 SQ = FIX(1/(2*PI*SQR(L1*IC1*CL/(C1+CL))))
9600 SQ = 0
10000 DF = FIX(10^INT(LOG(FA-FS)/LOG(10)-2)+5)
10100 DS = 0
10200 RETURN
10300 GOSUB 9700
10400 GOSUB 30300
PRINT
PRINT
PRINT
"FREQUENCY AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE"
10800 GOSUB 26600
10900 GOTO 6800
11000 IF N=3 THEN 10300 ELSE 11000
11100 REM
11200 REM ------------------ START-UP TIME CONSTANT ---------------------
11300 PRINT "THIS WILL TAKE SOME TIME"
11400 GOSUB 9700
11500 GOSUB 37700
11600 PRINT
11700 PRINT
11800 PRINT "FREQUENCY AT WHICH LOOP GAIN = 1 AT 0 DEGREES"
11900 GOSUB 26600
12000 PRINT "THIS YIELDS A START-UP TIME CONSTANT OF \"".CSNG(1000000/(2*PI*SG))\"" MICROSECS"
12100 GOTO 6800
12200 IF N=5 THEN PRINT ELSE 7300
12300 REM
12400 REM ------------------ STEADY-STATE ANALYSIS ---------------------
12500 PRINT "STEADY-STATE ANALYSIS"
12600 PRINT
12700 PRINT "SELECT 1 5031/5051"
12800 PRINT " 2 8751"
12900 PRINT " 3 5039/5040/5048/5049"
13000 PRINT " 4 8748/8749"
13100 INPUT ICX
13200 IF ICX<1 OR ICX>.4 THEN 12600
13300 GOSUB 46900
13400 GOTO 7300
13500 REM SUBROUTINE BELOW DEFINES INPUT-OUTPUT CURVE OF OSCILLATOR CKT
13600 IF ICX>2 AND VO=5 AND VI=2 THEN RETURN
13700 VO = -10*VI + 15
13800 IF VO>5 THEN VO = 5
13900 IF VO<2 THEN VO = 2
14000 IF ICX>2 AND VO=2 THEN VO = 5
14100 RETURN
14200 REM
14300 REM *************** DEFINE CIRCUIT PARAMETERS *********************
14400 REM
14500 REM
14600 REM
14700 INPUT " R1 (OHMS)\",R1
14800 INPUT " L1 (HENRY)\",L1
14900 INPUT " C1 (PF)\",X
15000 C1 = X*1E-12
15100 INPUT " CO (PF)\",X
15200 CO = X*1E-12
15300 INPUT " CXTAL1 (PF)\",X
15400 CX = X*1E-12
15500 INPUT " CXTAL2 (PF)\",X
15600 CY = X*1E-12
15700 INPUT "GAIN FACTOR MAGNITUDE", AV#
15800 INPUT "AMP FEEDBACK RESISTANCE (K-OHMS)", X
15900 RX = X*1000#
16000 INPUT "AMP OUTPUT RESISTANCE (K-OHMS)", X
16100 RO = X*1000#
16200 REM
16300 REM L(IS) CURRENT PARAMETER VALUES
16400 REM
16500 GO SUB 17100
16600 RETURN
16700 REM
16800 REM
16900 REM
17000 REM
17100 REM
17200 REM
17300 PRINT "CURRENT PARAMETER VALUES 1 RI = " , RI , " OHMS"
17500 PRINT " 2 L1 = " , CSG(L1) , " HENRY"
17600 PRINT " 3 C1 = " , CSG(C1*IE+12) , " PF"
17700 PRINT " 4 CO = " , CSG(C0*IE+12) , " PF"
17800 PRINT " 5 CXTAL1 = " , CSG(CX*IE+12) , " PF"
17900 PRINT " 6 CXTAL2 = " , CSG(CY*IE+12) , " PF"
18000 PRINT " 7 AMPLIFIER GAIN MAGNITUDE = " , AV#  
18100 PRINT " 8 FEEDBACK RESISTANCE = " , CSG(RX*001) , " K-OHMS"
18200 PRINT " 9 OUTPUT RESISTANCE = " , CSG(RO*001) , " K-OHMS"
18300 PRINT
18400 PRINT "TO CHANGE A PARAMETER VALUE, TYPE (PARAM NO ), (NEW VALUE) 
18500 PRINT "OTHERWISE, TYPE 0,0 
18600 INPUT N1,X
18700 IF N1=0 THEN RETURN
18800 IF N1=1 THEN RI = X
18900 IF N1=2 THEN L1 = X
19000 IF N1=3 THEN C1 = X*IE-12
19100 IF N1=4 THEN CO = X*IE-12
19200 IF N1=5 THEN CX = X*IE-12
19300 IF N1=6 THEN CY = X*IE-12
19400 IF N1=7 THEN AV# = X
19500 IF N1=8 THEN RX = X*1000#
19600 IF N1=9 THEN RO = X*1000#
19700 GOTO 17400
19800 REM
19900 REM
20000 REM
20100 REM
20200 REM
20300 REM
20400 REM
20500 REM
20600 REM
20700 REM
20800 REM
20900 REM
21000 REM
21100 REM
21200 REM
21300 REM
21400 REM
21500 REM
21600 REM
21700 REM
21800 REM
21900 REM
22000 REM
22100 REM
22200 REM
22300 REM
22400 REM
22500 REM
22600 REM
22700 REM
22800 REM
22900 REM
23000 REM
23100 REM
23200 REM
23300 REM
23400 REM
2251
REM 7 Amplifier gain in magnitude/phase form. AR+jAI = A at AP degrees

REM 8 (beta) in magnitude/phase form B pl+jBI = B at BP degrees

REM 9 Loop gain G = (BR+jBI)*(AR+jAI)

REM 10 Loop gain in magnitude/phase form GR+jGI = AL at AQ degrees

RETURN

REM The routine is entered with the starting frequency SQ+JFQ and
REM starting increment DS+JDF already defined by the calling program
REM In actual use either DS or DF is zero so the routine searches for
REM a GI=0 point by incrementing either SQ or FQ while holding the other
REM constant. It returns control to the calling program with the
REM incremented part of the frequency being within 1Hz of the actual
REM GI=0 point

23500 REM
23600 BR# = FNRR(RI, XI, (RI+RF), (XI+XF))
23700 BI# = FNRR(RI, XI, (RI+RF), (XI+XF))
23800 REM
23900 REM 7 Amplifier gain in magnitude/phase form. AR+jAI = A at AP degrees
24000 REM
24100 A = FNZM(AR#, AI#)
24200 AP = FNZP(AR#, AI#)
24300 REM
24400 REM 8 (beta) in magnitude/phase form B pl+jBI = B at BP degrees
24500 REM
24600 B = FNZM(AR#, BI#)
24700 BP = FNZP(AR#, BI#)
24800 REM
24900 REM 9 Loop gain G = (BR+jBI)*(AR+jAI)
25000 REM ~ G(real) + jG( imaginary)
25100 REM
25200 GR = FNRM(AR#, AI#, BR#, BI#)
25300 GI = FNXR(AR#, AI#, BR#, BI#)
25400 REM
25500 REM 10 Loop gain in magnitude/phase form GR+jGI = AL at AQ degrees
25600 REM
25700 AL = FNZM(GR, GI)
25800 AQ = FNZP(GR, GI)
25900 RETURN
26000 REM
26100 REM ****************************
26200 REM PRINT CIRCUIT ANALYSIS RESULTS
26300 REM
26400 REM
26500 PRINT " FREQUENCY = " , SG ,, " + J", FG ,, " Hz" 
26600 PRINT " XTAL IMPEDANCE = " , FNZM(RE, XE) ,, " OHMS AT " , FNZP(RE, XE) ,, " DEGREES"
26700 PRINT " (RE = " , CSNG(RE) ,, " OHMS"
26800 PRINT " (XE = " , CSNG(XE) ,, " OHMS"
26900 PRINT " LOAD IMPEDANCE = " , FNZM(RL, XL) ,, " OHMS AT " , FNZP(RL, XL) ,, " DEGREES"
27000 PRINT " AMPLIFIER GAIN = " , A ,, " AT " , AP ,, " DEGREES"
27100 PRINT " FEEDBACK RATIO = + , B ,, " AT " , BP ,, " DEGREES"
27200 PRINT " LOOP GAIN = " , AL ,, " AT " , AQ ,, " DEGREES"
27300 RETURN
27400 REM
27500 REM
27600 REM
27700 REM
27800 REM ****************************
27900 REM SEARCH FOR FREQUENCY (S+JF)
28000 REM AT WHICH LOOP GAIN HAS ZERO PHASE ANGLE
28100 REM
28200 REM
28300 REM This routine searches for the frequency at which the imaginary part
28400 REM of the loop gain is zero. The algorithm is as follows
28500 REM 1 Calculate the sign of the imaginary part of the loop gain (GI)
28600 REM 2 Increment the frequency
28700 REM 3 Calculate the sign of GI at the incremented frequency
28800 REM 4 If the sign of GI has not changed, go back to 2
28900 REM 5 If the sign of GI has changed, and this frequency is within
29000 REM 1Hz of the previous sign-change, exit the routine
29100 REM 6 Otherwise, divide the frequency increment by -10
29200 REM 7 Go back to 2
29300 REM The routine is entered with the starting frequency SQ+JFQ and
29400 REM starting increment DS+JDF already defined by the calling program
29500 REM In actual use either DS or DF is zero so the routine searches for
29600 REM a GI=0 point by incrementing either SQ or FQ while holding the other
29700 REM constant. It returns control to the calling program with the
29800 REM incremented part of the frequency being within 1Hz of the actual
29900 REM GI=0 point
30000 REM
30100 REM 1 Calculate the sign of the imaginary part of the loop gain (GI).
30200 REM
30300 GOSUB 20200
30400 GOSUB 26600
30500 IF GI=0 THEN RETURN
30600 6% = INT(SGN(GI))
30700 IF 6%=+1 THEN DS = -DS
30800 REM (Reversal of DS for GI=0 is for the pole-search routine.)
30900 REM
31000 REM 2 Increment the frequency
31100 REM
31200 SP = SG

22-52
31300 \texttt{FP = FQ}
31400 \texttt{SQ = SQ + DS}
31500 \texttt{FG = FQ + DF}
31600 \texttt{REM}
31700 \texttt{REM 3 CALCULATE THE SIGN OF GI AT THE INCREMENTED FREQUENCY}
31800 \texttt{REM}
31900 \texttt{GOSUB 20200}
32000 \texttt{GOSUB 2a600}
32100 \texttt{IF INT(SGN(GI))=0 THEN RETURN}
32200 \texttt{REM}
32300 \texttt{REM 4 IF THE SIGN OF GI HAS NOT CHANGED, GO BACK TO 2}
32400 \texttt{REM}
32500 \texttt{IF SQX+INT(SGN(GI))=0 THEN PRINT ELSE 31400}
32600 \texttt{SIX = -SQX}
32700 \texttt{REM}
32800 \texttt{REM 5 IF THE SIGN OF GI HAS CHANGED, AND IF THIS FREQUENCY IS WITHIN}
32900 \texttt{IH OF THE PREVIOUS SIGN-CHANGE, AND IF GI IS NEGATIVE, THEN}
33000 \texttt{EXIT THE ROUTINE (THE ADDITIONAL REQUIREMENT FOR NEGATIVE GI}
33100 \texttt{IS FOR THE POLE-SEARCH ROUTINE)
33200 \texttt{REM}
33300 \texttt{IF ABS(SP-SQ)<I AND ADS(FP-FQ)<I AND SQX=-1 THEN RETURN}
33400 \texttt{REM}
33500 \texttt{REM 6 DIVIDE THE FREQUENCY INCREMENT BY -10}
33600 \texttt{REM}
33700 \texttt{DS = -DS/10#}
33800 \texttt{DF = -DF/10#}
33900 \texttt{REM}
34000 \texttt{REM 7 GO BACK TO 2}
34100 \texttt{REM}
34200 \texttt{ODT0 31200}
34300 \texttt{REM}
34400 \texttt{REM}
34500 \texttt{REMEMber}
34600 \texttt{SEARCH FOR POLE FREQUENCY}
34700 \texttt{REM}
34800 \texttt{REM This routine searches for the frequency at which the loop gain = 1}
34900 \texttt{REM at 0 degrees. That frequency is the pole frequency of the closed-}
35000 \texttt{REM loop gain function. The pole frequency is a complex number, SQ+FG}
35100 \texttt{REM (Hz) Oscillator start-up ensues if SQ:0. The algorithm is based on}
35200 \texttt{REM the calculated behavior of the phase angle of the loop gain in the}
35300 \texttt{REM region of interest on the complex plane. The locus of points of zero}
35400 \texttt{REM phase angle crosses the j-axis at the oscillation frequency and at}
35500 \texttt{REM some higher frequency. In between these two crossings of the j-axis,}
35600 \texttt{REM the locus lies in Quadrant I of the complex plane, forming an}
35700 \texttt{REM approximate parabola which opens to the left. The basic plan is to}
35800 \texttt{REM follow the locus from where it crosses the j-axis at the oscillation}
35900 \texttt{REM frequency, into Quadrant I, and find the point on that locus where}
36000 \texttt{REM the loop gain has a magnitude of 1. The algorithm is as follows}
36100 \texttt{REM 1 FIND THE OSCILLATION FREQUENCY, 0+JFG}
36200 \texttt{REM 2 AT THIS FREQUENCY CALCULATE THE SIGN OF (AL-I) (AL = magnitude}
36300 \texttt{REM OF LOOP GAIN)}
36400 \texttt{REM 3 INCREMENT FG}
36500 \texttt{REM 4 FOR THIS VALUE OF FG, FIND THE VALUE OF SQ FOR WHICH THE LOOP}
36600 \texttt{REM GAIN HAS ZERO PHASE}
36700 \texttt{REM 5 FOR THIS VALUE OF SG+FG, CALCULATE THE SIGN OF (AL-I)}
36800 \texttt{REM 6 IF THE SIGN OF (AL-I) HAS NOT CHANGED, GO BACK TO 3}
36900 \texttt{REM 7 IF THE SIGN OF (AL-I) HAS CHANGED, AND THIS VALUE OF FQ IS}
37000 \texttt{REM WITHIN IH OF THE PREVIOUS SIGN-CHANGE, EXIT THE ROUTINE.}
37100 \texttt{REM 8 OTHERWISE, DIVIDE THE FG INCREMENT BY -10}
37200 \texttt{REM 9 GO BACK TO 3}
37300 \texttt{REM}
37400 \texttt{REM}
37500 \texttt{REM 1 FIND THE OSCILLATION FREQUENCY, 0+JFG}
37600 \texttt{REM}
37700 \texttt{GOSUB 9700}
37800 \texttt{GOSUB 30300}
37900 \texttt{REM}
38000 \texttt{REM 2 AT THIS FREQUENCY CALCULATE THE SIGN OF (AL-I)}
38100 \texttt{REM}
38200 \texttt{SYX = INT(SGN(AL-I))}
38300 \texttt{IF SYX=-1 THEN STOP}
38400 \texttt{REM ESTABLISH INITIAL INCREMENTATION VALUE FOR FG}
38500 \texttt{F1 = FG}
38600 \texttt{DF = (FA-F1)/10#}
38700 \texttt{GOSUB 30300}
38800 \texttt{DE = (FG-F1)/10#}
38900 \texttt{DF = 0}
39000 \texttt{FG = F1}
39100 REM 3 INCREMENT Fg
39200 REM Fg = Fg + DE
39400 REM 4. FOR THIS VALUE OF Fg, FIND THE VALUE OF SG FOR WHICH THE LOOP
39500 REM GAIN HAS ZERO PHASE. (THE ROUTINE WHICH DOES THAT NEEDS DF = 0.
39600 REM SO THAT IT CAN HOLD Fg CONSTANT, AND NEEDS AN INITIAL VALUE FOR
39800 REM DS, WHICH IS ARBITRARILY SET TO DS = 1000.)
40000 REM 40100 DS = 1000#
40200 SG = 0
40300 GOSUB 30300
40400 IF AL=1! THEN RETURN
40500 REM 40600 REM 5. FOR THIS VALUE OF SG+;Fg, CALCULATE THE SIGN OF (AL-1).
40700 REM 6. IF THE SIGN OF (AL-1) HAS NOT CHANGED, GO BACK TO 3
40800 REM 40900 IF SYX=INT(SQN(AL-1))=0 THEN PRINT ELSE 39400
41000 REM 41100 REM 7. IF THE SIGN OF (AL-1) HAS CHANGED. AND THIS VALUE OF Fg IS WITHIN
41200 REM 1HZ OF THE PREVIOUS SIGN-CHANGE. EXIT THE ROUTINE
41300 REM 41400 IF ABS(Fi-Fg)<1 THEN RETURN
41500 REM 41600 REM 8. DIVIDE THE Fg-INCREMENT BY -10.
41700 REM 41800 DE = -DE/10#
41900 REM P1 = Fg
42000 REM SYX = -SYX
42100 REM 42200 REM 9. GO BACK TO 3
42300 REM 42400 GOTO 39400
42500 REM 42600 REM 1. Compute approximate oscillation frequency
42700 REM 42800 REM The circuit model used in this analysis is similar to the one used
42900 REM in the small-signal analysis, but differs from it in two respects
43000 REM First, it includes clamping and clipping effects described in the
43100 REM text. Second, the voltage source in the Thevenin equivalent of the
43200 REM amplifier is controlled by the input voltage in accordance with an
43300 REM input-output curve defined elsewhere in the program.
43400 REM The analysis applies a sinusoidal input signal of arbitrary
43500 REM frequency, amplitude, at the oscillation frequency, to the XTAL1 pin, then
43600 REM calculates the resulting waveform from the voltage source Using
43700 REM standard Fourier techniques, the fundamental frequency component of
43800 REM the result is extracted This frequency component is then
43900 REM multiplied by the factor (ZL/(ZL+RO)), and the result is taken to be
4400 REM the signal appearing at the XTAL1 pin This signal is then
44100 REM multiplied by the feedback ratio (beta), and the result is taken to be
44200 REM the signal appearing at the XTAL2 pin The algorithm is now
44300 REM repeated using this computed XTAL1 signal as the assumed input
44400 REM sinusoid. Every time the algorithm is repeated, new values appear at
44500 REM XTAL1 and XTAL2. But the values change less and less with each
44600 REM repetition Eventually they stop changing. This is the steady state.
44700 REM The algorithm is as follows
44800 REM 45100 REM 1. Compute approximate oscillation frequency
45200 REM 2. Call a circuit analysis at this frequency.
45300 REM 3. Find the quiescent levels at XTAL1 and XTAL2 (to establish the
45400 REM beginning DC level at XTAL1)
45500 REM 4. Assume an initial amplitude for the XTAL1 signal
45600 REM 5. Correct the DC level at XTAL1 for clamping effects, if necessary
45700 REM 6. Using the appropriate input-output curve, extract a DC level and
45800 REM the fundamental frequency component (multiplying the latter by
45900 REM (ZL/(ZL+RO)))
46000 REM 7. Clip off the negative portion of this output signal, if the
46100 REM negative peak falls below zero
46200 REM 8. If this signal multiplied by (beta), differs from the input
46300 REM amplitude by less than 1mV, or if the algorithm has been repeated
46400 REM 10 times. exit the routine
46500 REM 9. Otherwise, multiply the XTAL2 amplitude by (beta) and feed it
46600 REM back to XTAL1, and go back to 5
46700 REM 46800 REM 1 COMPUTE APPROXIMATE OSCILLATION FREQUENCY
46900 GOSUB 9700
47000 REM
47100 REM 2. CALL A CIRCUIT ANALYSIS AT THIS FREQUENCY.
47200 GOSUB 20800
47300 PRINT PRINT PRINT "ASSUMED OSCILLATION FREQUENCY:"
47400 GOSUB 26600
47500 PRINT PRINT
47600 REM
47700 REM 3. FIND QUIESCENT POINT
47800 REM (At quiescence the voltages at XTAL1 and XTAL2 are equal. This
47900 REM voltage level is found by trial-and-error, based on the input-
48000 REM output curve, so that a person can change the input-output curve
48100 REM as desired without having to re-calculate the quiescent point.)
48200 VI = 0
48300 VB = 1
48400 K1 = 1
48500 Vi = VI + VB
48600 GOSUB 13600
48700 IF ABS(VO-VI)< .001 THEN 49200
48800 IF K1+SGN(VO-VI)=0 THEN 48900 ELSE 48500
48900 K1 = SGN(VO-VI)
49000 VB = -VB/10
49100 GOTO 48500
49200 VB = VI
49300 PRINT "QUIESCENT POINT = ";VB
49400 REM
49500 REM 4. ASSUME AN INITIAL AMPLITUDE FOR THE XTAL1 SIGNAL
49600 EI = 01
49700 NRX = 0
49800 REM
49900 REM 5. CORRECT FOR CLAMPING EFFECTS, IF NECESSARY.
50000 REM (K1 and K2 are curve-fitting parameters for the ROM parts.)
50100 K1 = (2.5-VO)/(3-VO)
50200 K2 = (VO-1.25)/(3-VO)
50300 IF IC%<2 OR IC%>4 THEN IF EI<0 THEN EO = VB ELSE EO = EI - 5
50400 IF IC%<1 OR IC%>3 THEN IF EI<(VB+5) THEN EO = VB ELSE EO = K1*EI+K2
50500 NRX = NRX + 1
50600 REM
50700 REM 6. DERIVE XTAL2 AMPLITUDE
50800 VO = 0
50900 VC = 0
51000 VS = 0
51100 FOR N% = -25 TO 24
51200 VI = EO - EI*COS(PI*N%/25)
51300 GOSUB 13600
51400 VO = VO + VO
51500 VC = VC + VO*COS(PI*N%/25)
51600 VS = VS + VO*SIN(PI*N%/25)
51700 NEXT N%
51800 VO = VO/50
51900 SI = ABS(VC^2+VS^2)/25*FNZM(RL, XL)/FNZM((RL+RO), XL)
52000 REM
52100 REM 7. CLIP XTAL2 SIGNAL
52200 IF VO-VI<0 THEN VL = 0 ELSE VL = VO-VI
52300 PRINT PRINT "XTAL1 SWING = ", EO-EI, " TO ", EO+EI
52400 PRINT "XTAL2 SWING = ", VL, " TO ", VO-VI
52500 REM
52600 REM 8. TEST FOR TERMINATION
52700 IF ABS(EI-VI)<.001 OR NRX=10 THEN RETURN
52800 REM
52900 REM 9. FEED BACK TO XTAL1 AND REPEAT
53000 EI = VI+B
53100 GOTO 50300
Design Considerations
When Using CHMOS
CHAPTER 23
DESIGN CONSIDERATIONS WHEN USING CHMOS

23.0 WHAT IS CHMOS?

CHMOS is Intel's n-well CMOS process which is based on the highly developed HCMOS-II technology. There are three other types of CMOS processes: p-well, twin-tub, and silicon on sapphire (SOS). All four CMOS structures are discussed in the accompanying article reprint, "Inside CMOS Technology." SOS and twin-tub offer superior performance, but are very costly to manufacture. The n-well technology offers about the same performance as p-well and has been chosen for Intel's microcontrollers because it is more readily adapted to the currently used and well understood HCMOS-II technology. This CMOS technology also offers a known path to higher performance products in the future.

Because CMOS tends to have lower gate density and a higher gate count than an NMOS circuit of the same functionality, the ability to scale down the transistor size in CMOS processes is essential to improving the price/performance ratio. The penalty paid for the size reduction, however, is a departure from the traditional CMOS supply voltage range of 3 to 18 volts. CHMOS will be limited to a maximum of 6 volts VCC.

Further reduction in CHMOS die size is accomplished by using dynamic nodes at appropriate points in the circuit, whereas, traditional CMOS is fully static. This reduces the gate count, and therefore, the die size — achieving lower cost. However, the use of dynamic nodes imposes a minimum clock frequency requirement on the CHMOS part.

23.1 NOISE IMMUNITY

CMOS noise immunity is greatly over stated. Noise immunity has been described as the amount of noise that can be induced at the input of a gate that will not change the logic state of the output. Noise margin, on the other hand, is the DC levels that will be applied to an input from another output (Voh, Vol) and the trip point of that input.

On the surface, CMOS would seem ideal for use in noisy environments. Output voltages are rail-to-rail, and input switch points are approximately 50% of VCC. There is one thing wrong with this analysis — CMOS has high impedance inputs. High impedance inputs need only voltage and very little current (nano Amps) to switch its output logic state. TTL, on the other hand, needs voltage and at least 20 μA (for an LS device) to switch its output logic state. Because it is a lot more difficult to induce noise in the form of current than in the form of voltage, CMOS tends to be more noise sensitive than TTL.

However, NMOS also has high impedance inputs and has input leakages typically less than 1 μA. Because NMOS output voltage swings are considerably less than CMOS, CMOS has the advantage when inputs and outputs are noisy. Another advantage of CMOS over NMOS is the p-channel pullup instead of a depletion pullup. The p-channel pullup is able to charge up the stray capacitance faster thus signal rise times are significantly improved.

In conclusion, don't fool yourself into thinking that CMOS eliminates the need to be concerned about noise problems. Time is well spent following good design practices and layout techniques from the earliest phases of a project.

23.2 LATCH-UP

CHMOS is not immune to traditional CMOS latch-up, but the latch-up threat is highly overrated. Latch-up is usually the result of unforeseen operating conditions, such as an unexpected power-up sequence, inadvertent removal and re-application of a supply voltage, or "hot-socketing" the part (plugging a chip into its socket while the system is active). An energetic voltage spike on VCC or the I/O lines might also trigger latch-up, but a little ringing on the data lines isn't going to cause any problems.

It is helpful to understand the mechanisms involved in the latch-up phenomenon. Figure 23-1A shows the circuit diagram of a typical CMOS output stage. Figure 23-1B shows a plan view of how this stage might look in n-well CMOS (with the gate electrodes of the FETs stripped away, since they are not germane to the discussion). There are two parasitic bipolar transistors in this structure, one pnp and the other npn. The n-well forms the base region of the pnp transistor, which is connected to VCC through the distributed resistance of the n-well. The source and drain of the p-channel pullup FET are dual emitters to the pnp transistor. The p-type substrate is the collector of this transistor and also serves as the base of the npn transistor which is connected to VSS through the distributed resistance of the substrate. The collector of this transistor is the n-well, and the drain and source of the pulldown n-channel FET are dual emitters. These parasitic transistors are shown in Figure 23-1C.

Any pullup FET that shares the same n-well region acts as additional multiple emitters to the parasitic pnp transistor. Much worse, ALL pulldown FETs and input protection devices on the IC act as additional collectors to the parasitic npn transistor. This has several implications. Latch-up is not limited to output stages, inputs only pins and internal gates can also be the cause. So when latch-up does occur, it's hard to tell which device caused it.

In normal operation, both of these parasitic bipolar junction transistors are in an off state, and do not hamper the operation of the FETs. However, if either parasitic junction transistor should turn on, its collector current might turn the other parasitic junction transistor on, and an SCR type effect rapidly ensues which is called latch-up.
What might turn on one of the parasitic junction transistors? Look at Figure 23-1C. The base of the parasitic pnp is at VCC, and the base of the npn is at VSS. If the output line swings a diode drop above VCC, or below VSS, it forward-biases the "D" emitter of either the pnp or the npn transistor.

Suppose it is the pnp emitter that is forward-biased by the output line swinging a diode drop above VCC. Some of the current that enters that emitter now exits the parasitic device as collector current. It flows down to the juncture of Rsub and the base of the npn transistor. If Rsub is low, the current is shunted through it to VSS, and the npn transistor stays off. If Rsub is high, some of the current crosses the base-emitter junction of the npn, and that transistor will likely turn on.

It is important to note that the designer of the integrated circuit has a certain amount of control over the turn-on probabilities. Grounded guard rings placed around the npn emitters will have the effect of reducing the value of Rsub.

Reducing the value of Rsub has the effect of increasing the amount of current that has to enter the pnp D emitter in order to turn the npn transistor on. The CHMOS, the npn transistor will not turn on if the current entering the pnp D emitter is less than 10 mA.

A similar sequence of events can occur when the output line swings a diode drop below VSS. In this case it is the npn transistor that is in danger of turning on the pnp, depending on the value of Rwell and how much current is involved. Again, the IC designer can reduce the value of Rwell by the judicious placement of guard rings, connected to VCC, around the pnp emitters. When the output line swings a diode drop below VSS, if the current that exits the output pin does not exceed 10 mA, there will be no latch-up.

### 23.3 POWER SUPPLY CONSIDERATIONS

The power supply, as viewed by the microcontroller, should be low in inductance because of the peak currents
associated with CHMOS switching characteristics. Note
that the repetition rate of these peak currents increases
with the clock frequency. Bypass capacitors of approxi-
mately 0.1μF should be used with proper PCB layout to
ensure a low inductance, high peak current power source.
Low inductance capacitors are also available that fit under
the package. These capacitors are also advantages for
HMOS in noisy environments (See Application Note
API125 “Designing Microcontroller Systems For Electrically
Noisy Environments” in this manual for detailed
discussion).

Power supply glitches must be filtered to ensure that the
maximum voltage rating of the device is not violated.
Violation may induce an SCR effect between VCC/VDD and
VSS (latch-up).

The polarity of the power supply must never be reversed.
The n-well is connected to VCC and the p-type substrate
is connected to ground so normally that pn junction is
reversed biased. If VCC or VDD are ever more negative
than −0.5 volts with respect to VSS, the n-well/substrate
pn junction will be forward biased and short VCC/VDD
to ground.

When the microcontroller is powered separately from the
surrounding circuitry, the microcontroller should always
be powered up before any input signal is applied. The
reverse is true when powering down, input signals first
then the microcontroller.

When separate VCC and VDD power supplies are used for
the 80C49, VDD and VCC must track each other
within 1.5 volts (except during power down) and also
maintain the 5v 20% specification. This ensures that the
CPU, powered by VCC, and the RAM, powered by VDD,
have proper voltage levels to communicate. If VCC and
VDD cannot be powered up simultaneously, VDD should
be applied first.

23.4 MINIMIZING POWER CONSUMPTION

The reason CMOS parts draw considerably less current
than an NMOS part is that there is no direct path between
VCC and ground (refer to figure 10-1A). When the
p-channel pullup FET is on the n-channel pulldown FET
is off and the output line gets pulled up. The opposite is
ture, when the n-channel FET is on the p-channel FET is
off and the output goes low. There is leakage associated
with either the n-channel and p-channel FETs in the off
state. The sum of all the leakages from every inverter on
the chip is what is called the "quiescent current." This
current is in the order of microAmps and is the lesser of
the two currents that add up to the total ICC.

When the inverter is switching in either direction, there
is a moment in time when both FETs are on creating a
low resistance path between VCC and ground. The ca-
capitance on the output line (pin capacitance, trace cap-
capitance, and input capacitance to the next stage) is also
charged or discharged which also increases current con-
sumption. This "dynamic current" is a couple of orders
of magnitude higher than the quiescent current.

Power supply voltage also comes into play with both types
of currents. If VCC is high, the leakage across the off
FET is increased. Output capacitance now has to be
charged up to a higher level. So a higher VCC also con-
tributes to making ICC larger.

Frequency of switch can also affect the amount of current
used. If the inverter stage is switched very slowly, then
the average current is equal to the quiescent current. As
the frequency of switching is increased, the dynamic cur-
rent contributes more and more to the total ICC until the
dynamic current totally swamps the quiescent current.

Now that we are aware of what components make up the
total ICC, we can work on ways of minimizing it. VCC
can play a big roll in minimizing power. If the whole
purpose in going to CMOS is to minimize heat dissipation
and the power is drawn from the 110 volt AC socket,
tweak your power supply to the minimum voltage that the
system will run. On the other hand if batteries are used
you must consider types of batteries available, lifetime of
the battery, and the voltage drop off at the end of its life
to determine what voltage to start at. Remember the lower
the voltage the lower the current draw.

The application also has to be analyzed for what kind of
response time is needed to determine how fast the micro-
controller needs to run. If the end application is a direct
human interface, then the response time can be relatively
slow and the processor can run at minimum speed. But if
real time decisions need to be made on evaluating incom-
ing data, then the microcontroller needs all the time it can
get. The idea is to run the microcontroller as slow as you
can and still get the task accomplished.

Intel has added idle mode and power down features to
help further manage your power consumption. Idle mode
in the 80C51/80C31 stops the clock to the CPU while
keeping the oscillator, RAM, interrupts, timer/counter,
and serial port alive. Stopping the clock to the CPU de-
creases the current in the CPU from dynamic to mostly
quiescent. Idle mode consumes approximately 1/100th the
operating current.

Idle mode allows the microcontroller to minimize its cur-
rent when no processing needs to be done. The live in-
terrupts, timer/counter, and the serial port can wake the
CPU and since the oscillator is running the response time
is quick.

Data on the ports are left in the state they were in when
idle was invoked. If careful attention is given to the logic
state of the port, total system current can be reduced. The
state of the port for reduced current is dependent on what
the port is driving. If it is a transistor, the transistor should
be put into its off state. If the port pin is driving another
CMOS gate, the loading of the port pin would be minimal
and a choice of the logic state may be made on what is
DESIGN CONSIDERATIONS WHEN USING CHMOS

Figure 23-2. MCS®-48 Power Down Circuit

on the outputs of the CMOS gate. If TTL is being driven, the reduced current would be when the port pin is high. A little thought in this area can go a long way in mini-
mizing system current.

Power down removes all internal clocks to decrease the current to totally quiescent current while the contents of the RAM are saved. This mode is useful in hand held applications where data needs to be saved between uses. On the 80C51, the port pins are left in the same state they were in when the power down mode was called. The same thought processes that were needed for reduced current in the idle mode are valid for power down also.

Figure 23-2 shows a simple circuit that uses one quad NOR CMOS integrated circuit and some external resistors and capacitors to power down the CHMOS MCS48 family under the control of one input. To activate the operation of the microcontroller, the AWAKE/ signal is pulled low and in turn, node A goes low turning on the p-channel VFET. This allows VCC to the microcontroller to be pulled to the power supply minus Vsd. Node A being low brings RESET high after a time determined by the R1C1 time constant. The RC has been chosen to allow 10 mS between VCC going high and RESET going high to allow the oscillator time to stabilize.

Powering down is accomplished by AWAKE/ going high which pulls RESET low. The time delay R2C2 allows the reset signal enough time to signal the microcontroller to save the RAM before VCC is shut off.

When idle mode and power down are used in conjunction with slow operating speed it can reduce your power needs to a minimum.

Unused input only pins on the MCS48 family such as SS/, T0, and T1 should not be allowed to float. The inputs would float about the trip point of the input buffer. This switching of the input buffer can waste up to .5 mA per input. Tie all unused input only pins high or low.

When the CHMOS units are being used with external program or data memory, PortO (Data Bus on the MCS48 products) is left floating when in idle mode. PortO also floats on the 80C51 when in the power down mode. The same condition as described above can happen with the input gate on those pins. Without tying these pins high or low at least .5 mA \times 8 or 4 mA could be wasted. Tie the PortO pins high or low through a 500K\Omega or 1M\Omega resistor. A little extra board space is minimal when considering the extra power savings.

The quasi-bidirectional pins have internal pullups so the inputs on these pins never float.

23.5 CHMOS I/O PORT STRUCTURE

The CHMOS I/O ports have similar drive capability to their HMOS counterparts, but the differences must be noted.

23.5.1 As An Output Pin

The I/O port structure is implemented as shown in Figure 23-3.

As an output pin latched to a low (0) state, pullups P1, P2, and P3 are in an off state while the pulldown N1 is on. This configuration uses little current, since there is no patch between VCC and ground in the output buffer stage.
When the output pin is to be latched in the high (1) state, the data line turns N1 off and turns on the weak (5μA) pullup P2. At the same time P1 (a strong pullup) turns on for one state time pulling the output up very quickly. Once the output voltage is above approximately 2 volts, P3 turns on to supply the source current. P3 and the inverter form a latch. This latch along with the support of P2 keeps the output high. Figure 23-4 shows the VOH vs IOH curve for this output structure when P1 is off.

### 23.5.2 As An Input Pin

To use the I/O port as an input, a one must be written to the pin first, leaving the pin in a high state. When the input goes below 2 volts, P3 turns off to avoid any high sink currents from being presented to the input device. Note when returning back to a one, P2 is the only internal pullup that is on. This will result in a long rise time if P2 is the only pullup.

### 23.5.3 Interfacing Between CHMOS and Other Logic Families

Interfacing Intel's CHMOS to other logic families is very simple and straightforward. When VCC is kept within 10% of 5 volts all inputs (except those noted in the data sheets) and outputs are TTL compatible. CMOS compatibility is achieved as long as VCC is kept within 20% of 5 volts.

When driving a high current load, the output current (Ioh) must be limited to keep the output voltage (Voh) at a minimum of 2 volts. If the voltage is pulled below 2 volts the output current will be dropped to approximately 5 μA. See Figure 23-4.
Inside CMOS Technology

Photo 1: The die for the 80C51, with the functions of the various sections identified.

How CMOS devices are manufactured and a look at three of them

by Martin B. Pawloski, Tony Moroyan, and Joe Altnether

Order Number: 230860-001
CMOS (complementary metal-oxide semiconductor) has often been called the ideal technology. It has low power dissipation, high immunity to power-supply noise, symmetric switching characteristics, and a large supply-voltage tolerance. But CMOS has rarely been used for advanced VLSI (very-large-scale-integration) microcomputer designs. Because of the complexity of the CMOS process, the ICs (integrated circuits) produced have traditionally had a relatively poor price/performance ratio.

As a result, CMOS was used only in applications that required low power and were neither performance conscious (such as in calculators and watches) nor cost conscious (many military applications, for example). Suddenly, however, all major semiconductor companies have announced either advanced CMOS products or the intention of designing their next generation of high-performance microprocessors using CMOS technology.

What has happened to make CMOS both affordable and high performance? For one thing, the dominant VLSI technology, NMOS (n-channel metal-oxide semiconductor), is rapidly approaching the process complexity of standard CMOS. It is not unusual nowadays for NMOS technology to have up to four transistor types with different operating characteristics. Much of the complexity of this process is added simply to help VLSI designers keep the operating power of their circuits under control.

Second, CMOS circuit designers are being more selective in the use of static CMOS logic. Critically placed dynamic logic, creative circuit design, and use of modes that offer varying degrees of power consumption are all tricks designers are using to maintain the advantages of CMOS.

Finally, aggressive reduction in CMOS transistor size is being used to bring CMOS performance in line with that of NMOS. As a matter of fact, many manufacturers are developing CMOS as a derivative of their advanced NMOS processes. This not only improves CMOS performance levels but also boosts reliability and reduces development costs.

The Evolution of LSI

Early LSI circuits were built with p-channel MOS transistors, which permitted high-circuit densities yet were relatively slow and difficult to interface to normal integrated circuits, such as TTL (transistor-transistor logic). As an example, the 1103-type 1K by 1-bit dynamic RAM (random-access read/write memory), circa 1971, required its inputs (address, controls, and data) to swing between 1 and 15 volts (V) although its output was measured in millivolts — hardly TTL compatible! About 1974, NMOS came to the rescue. It provided faster speed, and most of its inputs and outputs were TTL compatible.

Low power requirements are a major advantage of designing a system that uses CMOS.

NMOS was more difficult to manufacture than PMOS because contaminants would vary the thresholds of the n-channel transistors, causing deviations in speed and performance. But this problem was quickly overcome through ultraclean processing rooms, and NMOS became the workhorse technology because it cost less to manufacture, was easy to use, and had good speed-power characteristics. And NMOS technology had potential for greater improvement of its speed-power characteristics through scaling (or shrinking) of the silicon devices. The result of this scaling was HPMOS (high-speed NMOS), which accomplished three objectives: increased speed, reduced power, and increased density.

Over the past 10 years, the reduction in transistor size has, at the device level, increased memory density by a factor of 64, increased speed by a factor of 3, and reduced power consumption by a factor of 100. However, the scaling cannot continue ad infinitum because of resolution limitations of the photolithographic equipment used to make the circuits as well as breakdown mechanisms within the devices. More important, even before these limitations are reached, heat dissipation will prohibit major enhancements with NMOS. Heat generation increases exponentially with transistor count, and, at densities approaching 150,000 transistors per integrated circuit, special cooling measures are required. This heat can accelerate failure mechanisms within the silicon, reducing device and system reliability. To hurdle this barrier, low-power devices must be used.

The Importance of Power Consumption

The development of NMOS was spurred on by the semiconductor industry’s drive to produce high-volume, large-capacity memory devices, for which high density, rather than low power consumption, was the primary concern. As VLSI began to emerge, however, power dissipation became a limiting factor in continued increases in NMOS packing densities. Thus, the semiconductor industry turned to CMOS as a potential alternative.

CMOS achieves its low power dissipation through the use of both p- and n-channel transistors (hence the name “complementary”). Essentially, no DC power is dissipated in either logical state, and AC power occurs only during the relatively short switching period. Because most circuitry in a complex design is active only 10 to 20 percent of the time, CMOS achieves a dramatic reduction in power dissipation compared with NMOS, which continually dissipates DC power whenever an operating voltage is applied.

Low power requirements are a major advantage of designing a system that uses CMOS. Reducing power requirements has a domino effect that often substantially reduces the cost of the end product.
Figure 1: A comparison of NMOS and CMOS technologies. Figure 1a shows the schematic diagrams of an inverter as implemented in both NMOS and CMOS. A hypothetical input waveform and the resulting transistor currents are shown in 1b and 1c.

- low power allows smaller, lower-cost power supplies to be used
- power distribution in the system is simplified
- cooling fans can be eliminated
- printed-circuit boards can be packed more densely and can thus become smaller

With smaller power supplies, denser circuit boards, and no fans, smaller cabinets can be used, resulting in savings in chassis and enclosure costs. Also, power fail-safe and hand-held use become possible if battery operation is feasible.

Basic CMOS Operation
To truly understand the promises (and problems) facing both the CMOS VLSI digital designer and the CMOS systems designer, one must first understand some CMOS fundamentals.

Figure 1 compares the circuit diagrams and current characteristics of both an NMOS and a CMOS inverter. The NMOS inverter uses an n-channel depletion-mode transistor as the pull-up device (which drives the output line high) and an n-channel enhancement-mode transistor as the pull-down device (which drives the output line low). The pull-up transistor is used as a load; its operation approximates that of a constant current source. The pull-down transistor is used as the switching device; when active, it discharges the load, and when inactive it lets the pull-up charge the load. MOS loads are primarily capacitive and include the parasitic capacitances of the inverter itself, interconnect capacitances, and the thin-oxide capacitances of all the gates the inverter is driving.

Let’s note several characteristics of an NMOS inverter. When the pull-down device is turned on, it not only has to sink the current from the capacitive load, but it also has to sink the current supplied by the pull-up load device. Even in the quiescent state this current component from the pull-up device still exists. Because logic gates spend most of their time in the quiescent state, this quiescent current accounts for up to 90 percent of the total power dissipated in NMOS VLSI designs; the remaining 10 percent is switching or dynamic power.

A second related characteristic is that the inverter’s output voltage in the low state, $V_{OL}$, is dependent on the ratio of the impedances of the pull-down and pull-up devices. This ratio affects the noise margin and switching speed and is generally around 4:1. Such a ratio results in a $V_{OL}$ on the order of 0.2 V to 0.3 V. It also causes asymmetric switching characteristics: the fall time of the inverter is significantly faster than its rise time.

The CMOS inverter uses a p-channel enhancement-mode transistor as the pull-up device and an n-channel enhancement-mode transistor as the pull-down device. In a CMOS inverter, both the pull-up and pull-down transistors are used as switching devices. When the input changes from low to high, the p-channel device shuts off and the n-channel transistor discharges the load. When the input changes from
high to low, the n-channel device shuts off and the p-channel transistor charges the load. While almost all current from the CMOS inverter is used to charge or discharge the load, a small current component does not flow through the load. This is a result of the fact that both the p-channel and n-channel transistors are on for a short period of time during the input voltage transition. This current component is typically less than 10 percent of the total inverter current, though it depends greatly on the rise and fall times of the input signal.

With no quiescent power component, a CMOS inverter’s dynamic power dissipation represents only a small fraction of an equivalent NMOS inverter’s power dissipation. Also, the CMOS inverter is a “ratio-less” design, having only one transistor active after an input transition. This lets $V_{OL}$ go all the way to ground potential, resulting in better noise tolerance than NMOS inverters. It is also a simple matter to design CMOS circuits with outputs that have equal rise and fall times. While this is important in some circuits, it is generally not taken advantage of in VLSI designs because it requires greater chip area.

For NMOS and CMOS technologies with similar transistor dimensions and gate oxides, gate delays are essentially identical. The speed-power products for such a set of NMOS and CMOS technologies are shown in figure 2. This graphically illustrates the tremendous power advantage CMOS offers when used in high-performance VLSI designs.

While CMOS enjoys significant electrical advantages over NMOS, it does have a cost disadvantage. One small factor is the larger number of process steps needed to fabricate a CMOS device. More significant is the larger die required because CMOS has lower gate density.

**CMOS Technologies**

Figure 3 shows the four major CMOS technologies in use today: p-well bulk, n-well bulk, twin-tub bulk, and silicon-on-sapphire (SOS). P-well CMOS uses a p-type diffusion into an n-type bulk silicon substrate to form an n-channel transistor. The p-channel transistor is built directly in the bulk. This is the original CMOS technology, which has many years of good performance and reliability behind it.

The n-well CMOS process starts with a p-type substrate. N-type material is diffused into it to form the n-well in which p-channel devices are built. N-channel devices are built directly in the bulk substrate. An n-well CMOS process is usually derived from an advanced NMOS process. It also permits a highly optimized n-channel transistor, which yields a slight performance advantage over a p-well CMOS process.
Twin-tub CMOS combines n-well and p-well technologies by diffusing both an n-well for the p-channel transistor and a p-well for the n-channel transistor. The twin wells are usually formed in a lightly doped n-type substrate. While it is a slightly more complex and costly process than either n-well CMOS or p-well CMOS, twin-tub CMOS has the advantage of being able to optimize the performance of both the n-channel and p-channel devices. Thus, this process gives the highest overall performance of the bulk CMOS technologies.

The highest performing CMOS technology is SOS. Silicon islands are grown on an insulating sapphire substrate. N-channel or p-channel transistors are then built on the islands. High performance is achieved due to the significant reduction of parasitic capacitance. SOS also offers good gate density because no parasitic bipolar transistors are around to cause a phenomenon called latch up. Unfortunately, SOS devices are difficult and expensive to manufacture. For example, unused sapphire wafers cost approximately 10 times more than bulk silicon wafers.

While CMOS suffers a cost penalty of about 20 percent due to process differences, it generally suffers more significantly because of die size. (While processing steps have a linear relationship with cost, die size has an exponential relationship.) CMOS dies are larger than equivalent NMOS designs even when aggressive transistor scaling is employed. Three major factors contribute to this: the area used in trying to prevent latch up, CMOS logic-gate structure, and static design techniques.

**Latch Up Prevention**

Bulk CMOS technologies have parasitic bipolar transistors that, if improperly biased, can cause a phenomenon called latch up. This potentially destructive action results from triggering an SCR (silicon-controlled rectifier) formed by the transistors and can cause extremely large currents to flow. Figure 4 shows the construction of the parasitic SCR in an n-well bulk CMOS device.

![Figure 4: Parasitic SCR in bulk CMOS can cause latch up. Figure 4a shows how the parasitic transistors are formed in the silicon; figure 4b is a diagram of the equivalent circuit.](image)

Two well-defined conditions must exist before latch up can occur. First, for the SCR to be triggered, \( IR_{sset} \) must be greater than or equal to 0.7 V. This forward-biases the base-emitter junctions of the parasitic bipolar transistors. Second, to sustain the latch up condition, the product of the \( \beta \)s (gains) of the two bipolar transistors must equal at least 1.

In order to minimize the chance of one of the SCR's transistors being forward-biased, every attempt is made to reduce the resistance values as much as is feasible. This has the effect of requiring significantly larger injected currents before the SCR can be triggered. To reduce the resistance values, guard rings are used in the circuitry. (Guard rings are low-resistivity connections to the supply voltages placed around the CMOS p-channel and n-channel transistors.) While guard rings reduce the SCR bias resistor values, they also increase the space between n-channel transistors and p-channel transistors (thus reducing the gate density). To somewhat minimize this effect, particularly sensitive areas (like VLSI component's I/O pins) are heavily guarded, while the more protected internal circuitry is less so.

A less controllable method of preventing latch up is to try and decrease the \( \beta \)s of the parasitic transistors. While the vertical pnp transistor's \( \beta \) is set by the process design, the lateral npn transistor is more directly controllable. Its \( \beta \) can be drastically reduced by increasing the n-well-to-n+ diffusion spacing (or p-well-to-p+ diffusion spacing in p-well technology). This method reduces the \( \beta \) by increasing the width of the transistor's base. While this is an effective way of decreasing the gain of the parasitic structure, it also reduces the gate density.

In bulk CMOS technologies, to
give absolute protection against latch up is not only tremendously expensive in silicon area, but it is also virtually impossible. CMOS designers sacrifice area to ensure there is enough margin in their design to protect it from latch up in normal operating-system environments.

Logic-Gate Structures

Gate densities are also reduced in CMOS because standard CMOS logic gates are built from more transistors than their NMOS equivalents. Standard CMOS logic-gate design has a 1:1 ratio of n-channel transistors to p-channel transistors. For example, the two input gates shown in figure 5 take four transistors in CMOS and only three in NMOS. The relative density decreases as the number of inputs increases. For example, three-input gates require six transistors in CMOS and only five in NMOS, etc. As a matter of fact, it is rare to have a standard CMOS gate with more than three inputs because the self-loading and the transistor stack make the structure inefficient in both speed and area. On the other hand, it is not unusual in NMOS to have gates with as many as eight inputs.

Static Design Techniques

A final reason for the lower CMOS gate densities is the use of static logic (modern VLSI NMOS microcomputer designs rely heavily on dynamic circuitry). Dynamic circuitry essentially uses a small capacitor as a latch to store logic values. This technique saves both area (by reducing the number of transistors in a gate) and power (by reducing the number of gates in structures like latches, flip-flops, shift registers, etc.). Employing dynamic design can reduce an NMOS latch's area by 30 percent and its power consumption by 50 percent. However, the problem with dynamic circuitry is that the capacitor used to store the logic value is leaky and will, over time, discharge and lose its data. This is the same problem faced by dynamic memory designers. The solution is to periodically refresh the capacitor, which forces a minimum operating frequency to be adhered to.

CMOS can also use dynamic circuitry, especially to increase the ratio of n-channel transistors to p-channel transistors. Because static CMOS designs have a 1:1 ratio of n-channel to p-channel transistors, being able to increase this ratio will have the effect of giving CMOS a higher gate density (but the minimum operating-frequency characteristic of dynamic circuitry often conflicts with the CMOS potential of absolutely minimizing power). Therefore, while true static CMOS design does give the lowest possible power consumption (by allowing the device to operate at frequencies all the way to DC), dynamic CMOS designs, being more dense and resulting in smaller die sizes, tend to be more cost-effective. Thus, two trends are developing in the use of CMOS for VLSI microcomputer design.

Designers of the next generation of 16- and 32-bit microprocessors are choosing CMOS. Here, the goal is not to operate at the lowest possible power level but rather to keep the operating power under a maximum level for cooler junction temperatures, higher performance levels, and the ability to use standard low-cost packages. In these designs, extensive use is made of dynamic logic. The ratio of n-channel transistors to p-channel transistors is often as high as 3:1.

Designers of 4- and 8-bit single-chip microcomputers are choosing CMOS to accommodate a host of new portable, hand-held, and ultra-low-power applications. Here, the goal is to minimize the operating power levels consistent with the performance required by the application. In the simpler micro-
computers, true CMOS static logic is used—their simpler structure still allows a relatively small die size, while the low-performance applications they are appropriate for allow low operating frequencies. On the other hand, the more complex, higher-performance, single-chip VLSI components still make maximum use of static logic but are forced into dynamic logic for large arrays to keep the die cost down.

Future CMOS
CMOS will be the technology of choice for VLSI microcomputer designs. For one thing, with the advent of hundreds of thousands of transistors on a die, CMOS is the only technology that offers a cost-effective solution to the power-density problem.

A second and more subtle future issue is reduced supply voltage. As MOS transistors continue to be scaled to smaller dimensions to eke out further performance and density advances, the standard 5V supply voltage must be reduced, if only for internal circuitry, to limit substrate current and hot-electron effects. CMOS is better suited for lower supply-voltage operation because its switch point is a fixed percentage of the supply voltage. Also, due to its "ratio-less" structures, CMOS enjoys better noise tolerance than NMOS, another important factor at lower supply voltages.

Finally, CMOS has made and will continue to make major strides in its relative cost disadvantage to NMOS. Where CMOS formerly sold at as much as a fourfold premium, today it is selling at somewhat less than twice the price of comparable NMOS devices. With its continued use of standard, low-cost packaging technology as well as the more creative use of dynamic circuitry and hybrid static/dynamic designs, CMOS will rapidly approach the cost of NMOS.

As a matter of fact, several major semiconductor manufacturers have stated that CMOS/NMOS price parity will occur this decade, and some manufacturers say it will happen as early as 1985. When CMOS and NMOS cost the same, why would anyone buy NMOS?

---

A CMOS Single-Chip Computer: Intel's 80C51
by Martin B. Pawloski

Intel's 80C51 is an interesting example of how the static logic versus dynamic logic trade-off was made in an actual product design. The 8051 is an 8-bit, single-chip microcomputer with 4K bytes of ROM, 128 bytes of RAM, two 16-bit counter/timers, multilevel interrupt control, 32 I/O pins, full-duplex UART (universal asynchronous receiver/transmitter), and on-chip oscillator and clock circuits. A die with the sections identified by functions is shown in photo 1 on page 94.

The CMOS version of the 8051, called the 80C51, is targeted at a number of applications that require both high performance and low power consumption. In areas like telephony, automotive control, industrial control, and portable instrumentation, the 80C51 operates at or near its maximum speed, even if only for short intervals. (For example, most real-time applications need an external-interrupt response time of less than 100 microseconds (μs); more demanding applications require better than 10-μs response. While the response must be quick, and the interrupt routine executed quickly; the processor spends a significant portion of its time idle.)

Once the performance requirements of the application are known, it is possible to specify a minimum operating frequency. For the 80C51, a hybrid static/dynamic design was proposed that allows a minimum die size and includes various modes of operation to minimize power consumption.

First, the only areas of the design that were made dynamic were the (very large) ROM and Control arrays. These arrays contain almost 50,000 transistors and constitute a major portion of the die. By making them dynamic, an area savings on the order of 40 to 50 percent was accomplished.

Second, the processor, all the peripheral functions, the RAM, and the I/O ports were made static. This allowed two modes of operation other than normal operating mode: Idle mode and Power Down mode.

Because in many applications the processor does nothing more than wait for an event to happen, in the Idle mode the major clocks of the device are stopped and only smaller ancillary clocks operate to drive the peripheral counter/timers, external-interrupt control, and the serial channel. When one of the peripherals generates an interrupt, the processor

<table>
<thead>
<tr>
<th>Product</th>
<th>Technology</th>
<th>Vcc Supply Voltage Range</th>
<th>Operating Frequency</th>
<th>Normal Operating Mode (Icc Max)</th>
<th>Idle Mode (Icc Max)</th>
<th>Power Down Mode (Icc Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80C51</td>
<td>CMOS</td>
<td>4-6 V</td>
<td>12 MHz Max</td>
<td>24 mA</td>
<td>3 mA</td>
<td>50 μA</td>
</tr>
<tr>
<td>8051</td>
<td>HMOS</td>
<td>45-55 V</td>
<td>12 MHz Max</td>
<td>150 mA</td>
<td>—</td>
<td>20 mA</td>
</tr>
</tbody>
</table>

Table 1: A comparison of the CMOS and NMOS versions of the 8051.
clocks are restarted and instruction execution resumes in the interrupt-service routine. The Idle mode reduces power consumption by almost an order of magnitude.

In Power Down mode, all the clocks inside the device are shut off and only the internal 128 bytes of RAM are "kept alive." The only current consumed is a minute amount due to pn-junction leakage. Static logic was designed in the peripheral sections in order to support this mode because no clocks are available to refresh dynamic logic. In both the Idle and Power Down modes, special provisions are made for the dynamic circuits in the ROM and Control areas to enter a pseudostatic condition that prevents any extraneous power consumption due to voltage drift on capacitive storage nodes.

Table 1 compares the NMOS 8051 to the CMOS 80C51. The 80C51, designed in Intel's HMOS-derived n-well process called CHMOS, is less than 10 percent larger than the NMOS design and consumes only 15 percent of the normal operating power. More significant power savings are possible by operating the 80C51 at lower frequencies or by using the Idle mode.

A Look at CMOS Dynamic Memory
by Joe Altnether

The fast-growing portable-computer market is placing severe demands on semiconductor memory. For optimum system performance, these components must limit their power dissipation to suit battery operation and backup, and they must achieve the high data bandwidths and increased speeds needed for fast processing and high-resolution graphics. As the market reaches a projected $4.8 billion level by 1987 (a tenfold increase over 1982 levels), these requirements will combine to fuel the use of high-performance CMOS dynamic RAMs.

One architecture that can increase the speed of a CMOS dynamic RAM incorporates static-column address decoders: static circuits perform the selection of the column address of the RAM. Previously, this architecture has not been used with dynamic RAMs because of the increased power consumption of the static circuits over that of the dynamic circuits, and the advantage of low power consumption would have been lost. But with CMOS, the increased power consumption is negligible.

Memory-device Architecture

RAMs are organized internally as rows and columns of storage cells. Data access occurs at the intersection of a row address and a column address. In dynamic RAMs, the row and column addresses are multiplexed to reduce package size and pin count: the row addresses are clocked into the device with the RAS (row address strobe) signal, causing one row of data (1 bit from each of the 256 columns in a 64K-bit dynamic RAM) to be fed into the 256 internal sense amplifiers. (Because of the low internal signal levels, each column must have an associated sense amplifier to sense and restore memory-cell data.)

Next, column addresses are presented to the device and clocked into it with the CAS (column address strobe) signal. These column addresses are then decoded to select one of the 256 bits. Faster access and cycle times are obtained within a row (or "page") after the first access to it because the 256 bits within the row continue to reside in the sense amplifiers and need not be refetched. Re-applying only column addresses, then, in what is known as Page Mode operation, provides fast serial accesses and can increase cycle times by a factor of 2.

The CMOS dynamic RAM can incorporate static-column circuits to provide performance equivalent to that of high-speed static RAMs. With CMOS, the static-decoding circuits reduce the internal number of clocks by a factor of 3, eliminating the need to allow for setup and hold times of signals with respect to clocks and the need to compensate for timing skews due to process variances. With static-column circuits, precharge times are drastically reduced (in Page Mode operation of the Static-Column-mode device, precharge time is reduced from 30 nanoseconds [ns] to 5 ns). This precharge time reduction and the faster access times typically increase the memory's bandwidth to 20 MHz. (Performance of memory discussed here is based on the experimental 64K-bit CMOS dynamic RAM that Intel presented at the ISSCC conference in February 1983.)

With static-column architecture, two different types of Page Mode operation are possible: Static Column mode and Ripplemode. Static Column mode uses the RAS line and row addresses in the conventional manner, but once the row has been selected, data can be accessed merely by changing column addresses. As with a static RAM, column addresses must remain stable and valid for the entire address access cycle. Access time is measured from column addresses rather than the occurrence of CAS. (Typically, access from column addresses is 30 ns; from CAS, it is 10 ns.)

In operation, CAS is used to place the output in a high-impedance state or to activate an output buffer. CAS can be held active during the entire page cycle. In fact, it is possible to keep CAS permanently active (i.e., grounded). During a write cycle,
however, addresses as well as data are latched by CAS or WE, whichever occurs last. Operation is identical to that of an NMOS dynamic RAM in this case. This action ensures that the data is written into the proper memory location.

Although Static Column mode provides fast, easy accesses, speed at the system level is limited by how fast addresses for the next cycle become valid; the time to generate and stabilize the addresses must be added to the cycle time. Increased system speed can be obtained by using Ripplemode. With this mode, static-column circuits are again used to obtain access from valid column addresses, but the addresses are latched on the falling edge of CAS, removing the requirement for addresses to remain valid throughout the entire cycle. As a result, during the current cycle addresses for the next cycle can be set up or pipelined.

Column addresses enter the RAM through the internal address latch. This latch, controlled by CAS, provides flow-through operation. When CAS is inactive, the latch is open, and addresses pass through continuously to the static-column decoders. Any change in address is transmitted immediately to the decoder. Consequently, access to the RAM is again measured from valid column addresses. The latch captures the current address on the fall of CAS, permitting the system address to change while the access occurs. CAS also serves as an output enable on the data output. Static Column mode and Ripplemode both permit continuous data streams up to 20 MHz. CMOS technology and static-column architecture provide more than low power consumption and high bandwidth. In addition, static-column decoding simplifies system design by eliminating critical timing relationships while providing higher system speed. Access from column addresses gives usable speed for single random accesses within the RAM. Also, the CMOS technology enhances reliability by incorporating a mechanism to significantly reduce soft errors. Finally, increased stored charge creates larger internal signal levels, which can more easily be differentiated from noise. As a result, the CMOS dynamic RAM has wider operating margins and system reliability is improved.

**Power Consumption**

At the system level, dynamic memory has three components of power: active, standby, and refresh. The system's power consumption is defined as

\[ P = V(MI_a + KI_s + NI_f) \]

where \( P \) = system power, \( V \) = voltage (5.5 V worst case), \( I_a \) = active current, \( I_s \) = standby current, \( I_f \) = refresh current, \( M \) = number of active devices, \( K \) = number of devices in standby, and \( N \) = total number of devices. CMOS reduces the first term, the active current, relative to NMOS by a factor of 2. In addition, the lower active current reduces supply voltage transients, thus simplifying printed-circuit-board design and reducing decoupling-capacitor requirements.

The second term, standby current, is also reduced by a factor of 2 at TTL input levels. Driving the RAS signal to a CMOS level (\( V_{DD} = 0.5 \) V) places the device in a low-power-standby mode and typically draws 10 microamperes (\( \mu A \))—a factor of 50 reduction over NMOS!

Refresh current, the third term in the equation, is cycle-time dependent. Current increases with the frequency of refresh. In dynamic RAMs, data is stored on a capacitor that must be replenished or recharged every 2 or 4 milliseconds (ms). This refresh time is a function of the stored charge and the leakage current. With the CMOS dynamic RAM, the cell storage capacitance is 0.125 picofarad (pF) compared to 0.034 pF to 0.085 pF in an NMOS dynamic RAM. This low capacitance, coupled with lower leakage currents, permits the CMOS refresh period to be extended to 64 ms in standby.

At the standard 128 refresh cycles/2 ms (equivalent to a 15.625-\( \mu s \) refresh period), the NMOS device draws about 4.8 milliamperes (mA) and asymptotically approaches the standby current of 4 mA as the refresh period approaches infinity. Even eliminating refresh entirely only reduces the current to 4 mA, which is only a 16 percent improvement. As a result, extending NMOS refresh does not significantly reduce the system's power consumption.

Contrast this characteristic to the improvement CMOS offers. At 15.625 \( \mu s \), the CMOS dynamic RAM draws approximately 10 percent of the NMOS current, or 0.42 mA at TTL levels. Extending the refresh period reduces the current asymptotically to the standby current of 0.05 mA. At a 64-ms refresh period, the current is reduced to 0.15 mA, a 300 percent reduction. When battery powered, the CMOS system has a 10 times longer life than does the NMOS system, and an extended refresh mode offers another fivefold improvement. A 256-K-byte CMOS memory can retain data for nearly one week on only AA nickel-cadmium (nicad) cells—more than sufficient for most portable systems.

**High-Speed Applications**

Ripplemode and Static Column mode are ideal for applications involving high-speed buffers, telecommunications, and graphics. Bitmapped graphics systems would seem to be a natural fit with Page Mode operation. However, this was not always the case. Prior to the Intel 2164A 64K by 1-bit NMOS dynamic RAM, it was difficult to retrieve all 256 bits within a single row of memory because of the RAS-low time limitation of 10 \( \mu s \). Even with a Page Mode cycle time of 125 ns, to retrieve all 256 bits would require 32 \( \mu s \)—three times longer than allowed. The 2164A extended the RAS-low time to 75 \( \mu s \), permitting the extraction of all 256 bits during a single Page Mode cycle.

At the end of the cycle, the device cannot be reaccessed again until after a certain off-time allows internal nodes to be precharged to be ready for the next cycle. As a result, the 2164A can stream data at greater than...
a 7-MHz rate continuously. This function matches the timing and operation of low-performance, bit-mapped graphics memories. One 2164A, for example, can map all the data for the 256 by 256 matrix of a graphics display. During the horizontal scan time, the RAM performs a Page Mode cycle and one full line is displayed. During retrace time, the memory must be refreshed and can be updated with new data if required. This type of update is relatively slow; consequently, it limits the speed of animation on the screen because the processor has access to the memory only 25 percent of the time.

To increase resolution, more lines, each with more pixels, must be used. By performing two sequential Page Mode cycles from two different RAMs, pixel densities to 512 bits per line can be achieved. As pixel density increases, the memory cycle time must decrease to paint more pixels on a line in the same amount of time. This cycle-time limitation plus the fact that memory can be updated only during blanking has precluded dynamic RAMs from use in higher-resolution graphics displays. These systems are usually built with high-speed, expensive static RAMs.

With Ripplemode, memory update during screen display time, also known as cycle stealing, is possible. As an example, a 512 by 384 display requires 512 bits/line and 1 bit every 67 ns. Data is read from four memory devices in a series of eight Ripplemode reads each. Data is temporarily stored in a video-output register file and then shifted to the video screen at a rate slower than the Ripplemode reads. Following this, enough time is available to perform an update cycle before the next eight Ripplemode reads are performed to continue screen refresh. Eight was the number chosen to minimize the time the processor must wait to update the memory. In addition to this cycle stealing, which updates during display time, memory updates are also performed during blanking. Along with this system, a similar system was built using 2164As with Extended Page Mode operation. Each system used an iAPX 86 processor and similar software. A comparison of both systems showed the CHMOS (complementary high-speed metal-oxide semiconductor) system to have a 42 percent higher drawing speed. Animation on the CHMOS system was vastly improved.

**Usable Speed**

Memory design using dynamic RAMs has always been a challenge. Although multiplexing addresses does reduce the package pin count and increase system density, it limits the access and cycle times in the system. To access a dynamic RAM, low-order row addresses are presented and latched into the dynamic RAM with RAS. Row addresses must be held for a period \( t_{\text{RAS}} \) after the fall of RAS to guarantee proper operation. Next, the addresses must be changed to high-order column addresses and latched into the dynamic RAM with CAS, creating a timing window \( t_{\text{RCO}} \), which is the RAS-to-CAS delay.

Within this window, the designer must guarantee row address hold time, change the addresses, and account for any timing skew on the CAS signal. If column addresses are valid at the maximum specified \( t_{\text{RCO}} \), access time \( t_{\text{AC}} \) is measured from the high-to-low transition of RAS.

The cycle time is the sum of the access time and the cycle precharge time \( t_{\text{PR}} \). The access time is a function of \( t_{\text{RCO}} \), which has contradictory requirements. It must be as long as possible to simplify system design and at the same time as short as possible to enhance system speed. Cycle time is affected directly by the length of \( t_{\text{PR}} \).

Static-column operation eliminates the \( t_{\text{RCO}} \) problem. After row addresses have been latched into the RAM, the second portion of the access begins from valid column addresses. In other words, column access does not wait for CAS to become valid, but operates in a fashion similar to that of a static RAM. This is due to the flow-through operation of the CAS latch. CAS serves only to latch the addresses and to provide an output enable. Access from valid column addresses simplifies design by removing the CAS signal from the critical timing path.

Systems using dynamic RAMs are typically CAS access-limited because controllers generate timing signals in discrete clock increments. A CMOS dynamic RAM system might operate at 8 MHz without Wait states. Using any other 64K-bit dynamic RAM would require the injection of one or two Wait states, resulting in a corresponding performance penalty. Consequently, the advantage of higher processor speed is negated without the high-speed dynamic RAM. For systems incorporating either discrete or LSI controllers, the CMOS dynamic RAM simplifies the system design and offers higher system performance.

**High Reliability**

Soft errors are random, nonrecurring failures caused by ionizing radiation present within the environment. All matter contains small amounts of radioactive material. Alpha particles emitted by an IC's packaging material can penetrate the enclosed circuit. As they do so, they generate hole-electron pairs. Any high-impedance node in the vicinity sensitive to 1 million electrons may be affected, because the difference between a 1 and a 0 (known as the critical charge) is about 1 million electrons. Consequently, data in one cell could change from a 1 to a 0 or vice versa. Correct data can be rewritten into the affected cell and the memory will again function correctly, thus the term "soft error.

When first discovered during tests of 16K-bit dynamic RAMs, soft errors occurred at a rate five times greater than catastrophic or hard-error failures. While device designers worked to eliminate the alpha-particle sensitivity, systems designers added error-correcting circuits (ECC), which increased system reliability, but the systems were larger and more expensive due to the additional components required. Also, the system had to test and correct the data, slowing the system's performance. All this was due to soft errors. Obviously, what is really required is the elimination of soft errors.
CMOS technology offers such a solution. The CMOS dynamic RAM cell is built on an n-well in a p-substrate, creating a p-n junction or diode at the boundary. When alpha particles create hole-electron pairs in a CMOS device, something else occurs. First, the n-well is very shallow, and the majority of hole-electron pairs are created in the p-substrate. Holes cannot transfer across the reverse-biased p-n junction, which acts as a barrier to soft-error effects. Any electrons that do cross the junction are gathered at the +5V node away from the storage cell. The probability that sufficient hole-electron pairs are created within the n-well that cell upset could occur is so low that the soft-error rate of CMOS dynamic RAMs is typically orders of magnitude below that of their NMOS counterparts.

High storage capacitance also plays a role in the reduction of soft errors. The number of stored charged electrons representing a 1 or a 0 is directly proportional to the storage capacitance. Higher capacitance equates to more stored charge, which in turn increases the critical charge. The critical charge is the number of particles that differentiate a 1 from a 0. Increasing the critical charge beyond 1 million electrons significantly reduces the susceptibility to soft errors. This, in addition to the n-well mechanism, reduces the soft-error rate to much less than 0.001 percent per 1000 hours.

Studies were performed to compare reliability of systems with and without error correction for both NMOS and CMOS dynamic RAMs. The results show one surprise: at 256K bytes and below, the CMOS system without ECC is more reliable than the NMOS system with ECC, because of the cycle-time dependence of soft errors. In small systems, the memory is accessed more frequently, and the probability of a soft error is increased. With a soft-error rate at the very minimum 100 times less than NMOS, the CMOS dynamic RAM does not experience this effect.

Systems below 256K-byte capacities benefit by the elimination of ECC circuits from a cost, performance, and simplicity-of-design standpoint. First, ECC increases the access time of the system by 50 ns to check and correct data. Assuming a 120-ns RAM access, ECC increases the access by 42 percent. Moreover, the penalty on cycle time is even greater, especially when you are writing a single byte into a 2-byte word. In this instance, data must be accessed and corrected, the new byte merged into the word, and check bits generated. Finally, the system must write the new data into memory. Added to this are any system-timing skews. As a result, a 200-ns cycle time stretches to a 335-ns system cycle time or an increase of 68 percent. Therefore, using a CMOS dynamic RAM not only improves system reliability but enhances system speed and simplicity of design.

Joe Altnether is technical marketing manager at Intel Corp. (2111 N.E. 25th Ave., Hillsboro, OR 97123).
Modular Approach To C-MOS Technology Tailors Process To Application

KIM KOKKONEN AND RICHARD PASHLEY
Intel Corporation

ORDER NUMBER 231097-001
Modular approach to C-MOS technology tailors process to application

Despite the proliferation of applications, a few C-MOS process variations can address the functional requirements of many different products

by Kim Kokkonen and Richard Pashley, Intel Corp., Santa Clara, Calif

In the past few years, the interest in complementary-MOS technology and its applications to new products has exploded. Traditional arguments for C-MOS center on its low power dissipation, the large noise margins of complementary logic, and its simple ratioless design. With the advent of very large-scale integration, these arguments are taking on new meaning and importance.

As an example, Fig. 1 compares the performance of H-MOS (high-performance n-channel MOS) inverters with their equivalent in Intel's C-H-MOS (complementary high-performance MOS) technology. Though H-MOS's speed continues to improve with further scaling, its delay-power product is more than an order of magnitude higher than a C-H-MOS implementation with identical n-channel transistors. In a VLSI part with 50,000 gates, C-H-MOS could mean the difference between 1 and 10 watts of power dissipation, which might save the expense and difficulty of a sophisticated cooling system or extend a portable system's operating time by a factor of 10.

That C-MOS performance is now on a par with n-MOS technology has also accelerated its popularity. In addition, the density of C-MOS circuitry has improved dramatically with advances in technology. Finally, the number of process alternatives has grown so large that almost any integrated-circuit design can be supported with available C-MOS technology.

Unfortunately, the wave of enthusiasm for C-MOS and the needs of different applications have multiplied the number of approaches that C-MOS developers are taking. Several major issues remain in VLSI C-MOS design—namely latchup and soft-error prevention, interconnections, and logic-design techniques. A building-block approach with a limited number of basic process modules can be used to create a close-knit family of technologies that squarely addresses these issues and simultaneously supports a wide range of applications.

The basis for C-H-MOS

A firm foundation in n-MOS-transistor physics will support the advancement of C-MOS technology. As channel lengths approach 1 micrometer, n-channel transistors become more difficult to optimize because the standard 5-volt power supply causes problems with high-intensity fields. Improperly designed transistors may be unreliable as a result of hot-carrier injection into gate oxides, or they may cause less localized problems by injecting carriers into the MOS substrate—there to bleed charge from storage nodes or even trigger a destructive latchup.

The resources to develop and verify the reliability of a 1-μm n-channel transistor are well established and substantial. In Intel's C-H-MOS process, the basic design of the n-channel transistor is identical to its H-MOS counterpart, as shown in the table. Even at the more detailed levels of doping profiles, the H-MOS and C-H-MOS transistors are nearly identical.

Thus a high-performance C-MOS technology may be born out of an established n-MOS line. The relatively simple addition of an n-well in the same high-resistivity substrate results in a C-MOS process that serves as the basis for several optimized technologies. This is just a start, however, as other important issues remain.

Latchup has been the traditional nemesis of C-MOS. Given the presence of parasitic silicon controlled rectifiers within every bulk C-MOS chip, a current pulse of sufficient magnitude either inside or outside the chip may cause a catastrophic latchup. Many schemes have been proposed to combat latchup, ranging from carefully scrutinizing the layout (which imposes no burden on the technology) to a buried layer (which significantly in-
increases complexity and processing cost). All have some degree of effectiveness.

A modular approach to a broad-based line of C-MOS technologies requires other measures besides mere physical latchup resistance. The latchup spoiler must be applicable to dynamic random-access memories, erasable programmable ready-only memories, and static RAMs, as well as to microprocessors and controllers. In order to improve latchup resistance, it cannot increase the distance between n- and p-channel transistors (this constraint is most significant in random logic and full C-MOS six-transistor static-RAM cells). The technique must be compatible with low-cost and large-volume manufacturing. Finally, the approach must be consistent with the use of an automated checking algorithm, so that every gate of a large semirandom logic design need not be scrutinized for latchup susceptibility.

**Epitaxial benefits**

Figure 2 shows the margin gained in latchup trigger current when an epitaxial substrate is used. The epitaxial substrate brings the same latchup benefits to all product lines, and in many cases provides additional advantages such as improved surface lifetimes (for dynamic RAMs) and reduced dc resistance (for E-PROMs and logic). Epitaxial substrates are now available in volume from commercial silicon vendors, adding less than 5% to the cost of a finished wafer. No additional or exotic fabrication equipment needs to be installed. Because the epitaxial substrate's heavily doped bulk effectively eliminates the vertically triggered latchup mode, it is possible to develop a set of computer-aided-design tools that can flag latchup-sensitive layouts on the largest VLSI chips.

Since grasping the phenomenon of upsets induced by alpha particles, in 1977, memory designers have taken care to ensure that enough charge is stored within each cell to minimize the problem. As critical chip dimensions are reduced, this problem becomes more severe, however, since both parasitic and storage capacitances are naturally reduced. For the latest 1.5-µm n-MOS process, stored charge is low enough to caution even microprocessor designers to guard against random storage nodes suffering from soft errors. Fortunately, C-MOS provides a natural barrier against soft errors if the storage node is located within the C-MOS well.

The well junction is reverse-biased by the power-supply voltage. The electric field at this junction naturally repels any carriers generated outside the well that might otherwise diffuse up to surface storage nodes. The combination of the well structure and an epitaxial substrate is even more effective. Here the funneling mechanism that usually collapses local electric fields during the passage of an alpha particle is also minimized. By using epitaxial substrates and the protection of a C-MOS well, the amount of charge collected during an alpha event can be reduced by an order of magnitude.

Of course, the designer must arrange for the storage node to reside within the well. This constraint, combined with other performance issues, leads to different choices of well and substrate polarities, depending on application. For example, in a C-MOS technology that is optimized for dynamic RAM, the ideal memory cell should have a p-channel pass gate and a p-channel capacitor located within an n-well in a p-type substrate. The p-channel transistor is chosen because it injects far fewer spurious carriers into the substrate and thus does not by itself disturb the state of neighboring cells.

The conductance of the p-channel device, while lower than that of an n-channel device of the same size, does not degrade the RAM's performance, since dynamic-RAM sensing is limited primarily by the amount of stored charge. Experimental results with C-H-MOS dynamic RAMs based on these principles show a soft-error rate of less than 300 FIT (failures in time, or device failures per billion hours) at a power supply of only 3 V. This is an improvement of more than three orders of magnitude over traditional n-MOS dynamic-RAM technology and offers the possibility of dynamic-RAM systems that require no error correction and that are compatible with low-voltage battery backup.

High-density, high-performance static RAMs present the other side of the coin. The smallest static-RAM cells today are built using polysilicon-load resistors that sustain the stored-node voltage. On the time scale of an alpha event, however, these resistors in effect do not exist. Because the storage node's RC time constant is on the order of milliseconds and the alpha event's time scale...
is nanoseconds, the cell appears dynamic. In this sense, polysilicon-load static-RAM cells are very similar to dynamic-RAM cells. The major difference arises in the way the cells sense the cell's information. The static-RAM cell provides a direct current, and to maximize the cell's performance, that current must be as large as possible while contained in a minimum area. Thus the chip designer must use high-gain n-channel transistors for the cell's pass gates and pulldowns. For good soft-error protection, then, the cell must be located in a p-well within an n-type substrate.

The p-well approach benefits even full CMOS six-transistor static-RAM cells. The area of such cells depends strongly on the distance allowed between p- and n-channel devices. Using a straightforward implementation of epitaxial CMOS, the p-well approach provides more margin against latchup at small n-to-p spacings (Fig. 3). This phenomenon occurs because of the differing diffusion properties of n- and p-type dopants. The heavy doping in the n-type substrate is less mobile than is the p-type dopant, resulting in less outdiffusion during thermal processing and thus minimizing the shunt resistance that controls latchup.

Hooking it up

One of the challenges of CMOS in logic applications is interconnection. Designers of n-MOS chips are accustomed to buried contacts, which directly connect n-type polysilicon and n-type transistor source or drain regions. Because CMOS requires contact to both p and n regions, the traditional n-type buried contact becomes much less useful, and a version suitable for both diffusion polarities is quite difficult to implement. This increases the burden on contact and metallization modules.

For high-density CMOS logic, the first level of metal is all but consumed by local connections between p and n transistors. The payback from adding a second level of metal for longer-distance routing is very high. A good example exists for the six-transistor static-RAM cell commonly used by logic designers. Figure 4 compares single- and double-metal versions of this cell, both implemented with 1.5-μm design rules. Here the second-metal layer provides the bit lines for the cell. Similar arguments justify the use of second metal in global power, clock, and data routing in complex microprocessor chips.

Contacts themselves are more difficult to build in CMOS. N-MOS technology accustomed process engineers to adding a phosphorus contact plug after the contacts have been etched. This plug brought several advantages: the phosphorus gettered metallic contaminants from the wafer, reducing junction leakage; and the high-temperature diffusion rounded the profile of the contact sidewall, easing the step coverage of the metal subsequently deposited. Further, the plug had self-aligning features. If the contact etch attacked the silicon substrate or if the contact was misaligned toward the field-oxide edge, the plug would rejuvenate the resulting weakened junctions. In CMOS, these same attributes must be obtained differently, through improved fabrication, cleanliness, new gettering techniques, improved dielectrics, and tightly controlled contact etching. Figure 5 shows the difference in implementing a 1.5-μm contact structure in n-MOS and CMOS. Along with the proliferation of CMOS technologies has come a wave of innovation in CMOS design techniques. For digital logic, the major contenders for broad use are full complementary design and domino logic, first proposed by AT&T Bell Laboratories (Fig. 6). For many applications, traditional CMOS logic is a winner. It requires no clocks, has larger operating margins, and uses fewer transistors for simple gates. For more complex gates, however, domino logic uses fewer transistors and runs faster. The speed results from connecting fewer transistors in series and reducing gate-fanout loading by

Electronics / May 3, 1984

23-20
up to a factor of two compared with full C-MOS.

Interestingly, the choice of design style influences the optimal type of C-MOS well. The speed of full C-MOS is limited by the slower of the two transistor polarities. Since the trip point is quite close to half the power supply, the time required for either transistor type to discharge its load capacitance by about 2.5 V sets the gate’s speed. Since the p-channel device is the weaker one, it pays to choose a well type that improves the p-channel’s conductance. P-well does this because the p-device is fabricated in an uncompensated substrate and thus has maximum mobility. Comparisons between n-

5. Making contact. Contacts are more difficult to build in C-MOS than in conventional n-MOS. The phosphorous contact plug used in n-MOS after contact etching (a) adds desirable features such as reduced junction leakage and improved step coverage by the metal layer. To gain the same advantages in C-MOS requires greater process control (b) and p-well construction show that the p-channel’s conductance may be improved by as much as 10% with the proper well type.

By contrast, domino logic is at its best in an n-well technology. Here, the n-channel transistor dominates both performance and transistor count. Placing the n-channel device outside the well improves its conductance and reduces the dominant parasitic junction capacitance. Density also increases because no well contacts are required for the majority of the transistors.

The twin-well approach to C-MOS blurs these distinctions. In this approach, a high-resistivity epitaxial layer is grown on a heavily doped starting wafer. Then the doping for each transistor polarity may be independently optimized without need for doping compensation. Performance arguments based on mobility or junction capacitance thus become moot. Nonetheless, domino logic will still be best on a p-type substrate (equivalent to n-well) because it does not require well contacts to collect the large parasitic substrate currents from the n-channel transistors, thus improving packing density.

Matching process to product

These and other technical arguments may be combined into a consistent strategy (Fig. 7) for creating a line of C-MOS processes serving a broad marketplace. For at least the next several years, a complete technology line must include C-MOS based on both p- and n-type substrates. Fortunately, choosing epitaxial-latchup control minimizes the development cost of running both process-

6. Logic. Two major contenders for digital logic design are full complementary (a) and domino logic (b). The former requires no clocks and is simpler for many applications. Domino logic, which performs best in an n-well technology, is faster and simpler for more complex circuits.
es. Dynamic RAMs are supported on the n-well side to minimize pattern sensitivities induced by substrate currents while protecting the p-channel cell from soft errors. E-PROMs are built in a similar n-well C-H-MOS process. Placing Intel's n-MOS E-PROM cell in an epitaxial p-type substrate eliminates parasitic effects caused by high substrate currents flowing during cell programming.

Microcontrollers land on the n-well side also, so that they may incorporate on-chip E-PROM cells. Most microcontroller products come in two versions, one with on-chip E-PROM for system-development and manufacturing flexibility, and another with on-chip ROM for lowest cost. Using n-well C-MOS, a single core design can support both versions. Telecommunications and signal-processing products can also take advantage of the n-well E-PROM process, both for its high-quality polysilicon-polysilicon capacitors and for the E-PROM cell's programmable features. High-performance static RAMs, whether six-transistor or polysilicon-load, can take advantage of a p-well C-H-MOS process. High-end microprocessors can key off the dense n-to-p packing and double-metal capability offered by the six-transistor static-RAM process.

Because these processes are modular, development is simplified and manufacturing overhead is minimized. Just as all the 1.5-µm C-H-MOS III technologies share a common transistor module, the difficult contact module was developed once to be shared among all. Specialized features such as double polysilicon or double metal are extensions of the common base.

The future

C-MOS technology is still developing at a frenetic pace. Surprisingly, the application of some newer techniques and the demands of next-generation circuits may bring the various forms of C-MOS closer together, rather than further splitting the number of integrated processes.

One example of this trend is the development of a trench-isolation technique for separating n- and p-devices. When this module is perfected, there will be no reason to develop six-transistor static-RAM cells on p-well technology. The near-ideal trench isolation will prevent latchup on either substrate type. Similarly, if stacked C-MOS static-RAM cells can be perfected, there will be no need for polysilicon loads. The stacked C-MOS cell will have the same density but with improved performance and soft-error immunity. At that time, twin-well C-MOS on a p-type substrate, augmented by specialized features for specific product lines, will become the one approach to a broad line of C-MOS processes.

Another factor affecting future C-MOS integration is the continued scaling of transistors. It is well known that the weaker p-channel transistor is gradually catching up on the n-channel device as channel lengths enter the submicrometer region. Eventually, the performance differences may become so small that p- and n-channel devices will be used interchangeably. Before this level is reached, however, the 5-V power-supply standard must be reduced. Because of the large base of TTL-compatible designs and the impossibility of converting the world to a new standard overnight, components operating from the new reduced supply will need to maintain TTL compatibility and also be able to operate in a system that mixes older 5-V components with lower-voltage ones. An on-chip 5-to-3-V converter may be one way to solve the problem. This technique, however, will waste up to 40% of the total chip power within the voltage regulator.

C-MOS technology provides an elegant solution because it can drive TTL-compatible output levels from a system power supply as low as 3 V. Since TTL levels are referenced to the negative (ground) rail, the grounded substrate offered by n-well C-MOS is a much-preferred means of integrating submicrometer transistors into such a system. This will be a strong motive to standardize on p-substrate C-MOS.

A final factor that tends to drive future C-MOS processes toward commonality is the growing importance of RC delays in overall chip performance. The latest high-performance static RAMs use an aluminum strap in parallel with the polysilicon word line because the RC delay induced by even the best refractory metal polycides is several nanoseconds too long. Studies of dynamic RAMs larger than 1 megabit similarly indicate that refractory word lines will probably be inadequate, forcing the technology to support two layers of metal. Combining these observations with those made previously regarding the evolution of static-RAM cells leads to the conclusion that most future C-MOS technologies will have two layers of polysilicon as well as two layers of metal.

The development of silicon-on-insulator technology is the one major factor that could renew the divergence of C-MOS approaches in the future. However, until the quality of SOI substrates is adequate to support dynamic RAM and E-PROM cells, and not just static logic, it will not play a major role in a broad-based and modular technology strategy.

Electronics / May 3, 1984

23-22
CHAPTER 24
ADVANCED PACKAGING

24.1 Introduction

Today, system designers using LSI and VLSI devices are continuously facing problems associated with achieving the highest system performance level and most complex functional level of a particular system application in the smallest physical size possible. Until recently the available solutions to these device problems were limited to the traditional standard dual-inline-package (DIP) based on 100mil center-to-center lead spacing and flat packages (FP). Today, these device problems are being solved in a number of new ways; DIPs based on 50 mil center-to-center lead spacing, surface mounted small outline DIP, surface mounted chip carriers, surface mounted gull-wing flat package and pin grid arrays. Among these possible solutions the two that are emerging as the next standard IC packages for LSI/VLSI devices are the surface mounted chip carrier and the pin grid array.

24.2 What Are Surface Mounted Chip Carriers?

The chip carrier is basically the business portion of a DIP. Chip carriers are available in two general types: leadless and leaded.

The leadless chip carrier construction is accomplished in much the same manner as the multi-layer ceramic DIP package, but it is missing the side-brazed legs and much of the ceramic surrounding the die cavity area. Instead, it consists of a ceramic package with I/O pads on all four sides, a die cavity area, metalization traces to the I/O pads, and a hermetically sealable lid. Leadless chip carriers are available in either square or rectangular package outlines. The leadless chip carrier can be attached to a board surface either directly, by socketing, or by the addition of add-on leads.

The leaded chip carrier construction is accomplished also in much the same manner as the plastic DIP package, but has leads that are bent down and under the package on all four sides rather than like the DIP. It is also missing much of the plastic surrounding the die platform and consists of plastic material encapsulating the die platform with I/O pads or leads on all four sides. The plastic leaded chip carrier is available in square or rectangular package outline, that can be attached to a board surface either directly or by a socket.

Chip carriers are registered JEDEC standard packages. The standard is based upon two basic package types, one with 50mil center-to-center terminal spacing, the other with 40mil spacing. Each package type was developed with certain application in mind, such as mounting methods, board material, thermal characteristics and external features. Intel’s Microcontroller Operation offers two varieties of JEDEC packages, both with 50mil spacing, a square ceramic leadless type and a square plastic leaded type.

In addition to these mechanical packaging advantages, there are also electrical benefits. The package arrangement of the chip carriers I/O pads, on all four sides, allows for package traces to be shorter and more uniform in length. This allows lower resistance, less capacitance and less inductance, resulting in higher system performance and improved switching characteristics.

24.3 Why Chip Carriers?

Figure 24.1 shows the differences between the surface area (in²) versus pin count of both a ceramic dip and a ceramic leadless chip carrier device. Note that an 18-pad chip carrier offers a 57% saving in area. As the pin counts increase, the chip carrier surface area advantage becomes significantly more obvious. This space efficiency allows the system designer to increase the number of components on a board or decrease the overall board size and, thus, the overall system size.

![Figure 24.1. Package Area](image-url)

Figure 24.1. Package Area
For an 18-pin count, there is a 77% weight saving and as the pin counts increase to 28 and to 48, this weight savings increases to 90% and 95%, respectively. In the case of the plastic leaded chip carrier, the weight savings over a plastic DIP is noticeable but not significant. However, it is in the ability to decrease the board size and, thus, economizing on material and weight reduction that the significant advantage exists.

### 24.4 What Are Pin Grid Arrays?

The Pin Grid Array is basically a combination of the ceramic DIP and ceramic Leadless Chip Carrier. The Pin Grid Array construction is accomplished in much the same manner as the multi-layer ceramic DIP, but it is missing the side-brazed legs and much of the ceramic surrounding the die cavity. Instead, it consists of a ceramic package with leads coming off the bottom in rows or circular patterns. The Pin Grid Array is available in square package outlines with lead spacing of 100 mils and can be attached to a board in the same manner as a DIP.

Pin Grid Arrays are being proposed as JEDEC standard packages and will have from 1 to 10 nested rows of legs and may have a die cavity mounting area oriented up or down.

### 24.5 Why Pin Grid Array?

Figure 24.4 shows the difference between the surface area (in\(^2\)) versus pin count of both a 50 mil spacing chip carrier and a 100 mil spacing Pin Grid Array device. Note that at approximately 68 pins and above the Pin Grid Array becomes a better solution for higher pin count requirements than the chip carrier. In addition, the 100 mil lead spacing and through board mounting technique provides customers with an assembly technology that is familiar.
CERAMIC LEADLESS CHIP CARRIER
All dimensions in inches and (millimeters)

44-Leadless Hermetic Chip Carrier
JEDEC Package Type C
CERAMIC LEADLESS CHIP CARRIER
All dimensions in inches and (millimeters)

44-Leadless Hermetic Chip Carrier
JEDEC Package Type C
PLASTIC LEADED CHIP CARRIER
All dimensions in inches and (millimeters)

44-Leaded Chip Carrier
JEDEC Package Type C
Pinouts
MCS®-48 Family

80C49/39*

8749H*

*Top View Looking Through Package
CERAMIC PIN GRID ARRAY
All dimensions in inches and (millimeters)

68-Pin Hermetic Pin Grid Array
Pinouts
MCS®-96 Family

MCS®-96 Pin Table

<table>
<thead>
<tr>
<th>PIN</th>
<th>SYMBOL</th>
<th>PIN</th>
<th>SYMBOL</th>
<th>PIN</th>
<th>SYMBOL</th>
<th>PIN</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACH7/P0.7</td>
<td>18</td>
<td>AD0/P3.0</td>
<td>35</td>
<td>READY</td>
<td>52</td>
<td>HS1/HSO4</td>
</tr>
<tr>
<td>2</td>
<td>ACH6/P0.6</td>
<td>19</td>
<td>AD1/P3.1</td>
<td>36</td>
<td>T2RST/P2.4</td>
<td>53</td>
<td>HS10</td>
</tr>
<tr>
<td>3</td>
<td>ACH2/P0.2</td>
<td>20</td>
<td>AD2/P3.2</td>
<td>37</td>
<td>BHE</td>
<td>54</td>
<td>HSIO</td>
</tr>
<tr>
<td>4</td>
<td>ACH0/P0.0</td>
<td>21</td>
<td>AD3/P3.3</td>
<td>38</td>
<td>WR</td>
<td>55</td>
<td>P1.4</td>
</tr>
<tr>
<td>5</td>
<td>ACH1/P0.1</td>
<td>22</td>
<td>AD4/P3.4</td>
<td>39</td>
<td>PWM/P2.5</td>
<td>56</td>
<td>P1.3</td>
</tr>
<tr>
<td>6</td>
<td>ACH3/P0.3</td>
<td>23</td>
<td>AD5/P3.5</td>
<td>40</td>
<td>P2.7</td>
<td>57</td>
<td>P1.2</td>
</tr>
<tr>
<td>7</td>
<td>MMI</td>
<td>24</td>
<td>AD6/P3.6</td>
<td>41</td>
<td>VBB</td>
<td>58</td>
<td>P1.1</td>
</tr>
<tr>
<td>8</td>
<td>EA</td>
<td>25</td>
<td>AD7/P3.7</td>
<td>42</td>
<td>VSS</td>
<td>59</td>
<td>P1.0</td>
</tr>
<tr>
<td>9</td>
<td>VCC</td>
<td>26</td>
<td>AD8/P4.0</td>
<td>43</td>
<td>HSO3</td>
<td>60</td>
<td>TXD/P2.0</td>
</tr>
<tr>
<td>10</td>
<td>VSS</td>
<td>27</td>
<td>AD9/P4.1</td>
<td>44</td>
<td>HSO2</td>
<td>61</td>
<td>RXD/P2.1</td>
</tr>
<tr>
<td>11</td>
<td>XTAL1</td>
<td>28</td>
<td>AD10/P4.2</td>
<td>45</td>
<td>P2.6</td>
<td>62</td>
<td>RESET</td>
</tr>
<tr>
<td>12</td>
<td>XTAL2</td>
<td>29</td>
<td>AD11/P4.3</td>
<td>46</td>
<td>P1.7</td>
<td>63</td>
<td>EXTINT/P2.2</td>
</tr>
<tr>
<td>13</td>
<td>CLKOUT</td>
<td>30</td>
<td>AD12/P4.4</td>
<td>47</td>
<td>P1.6</td>
<td>64</td>
<td>VPD</td>
</tr>
<tr>
<td>14</td>
<td>TEST</td>
<td>31</td>
<td>AD13/P4.5</td>
<td>48</td>
<td>P1.5</td>
<td>65</td>
<td>VREF</td>
</tr>
<tr>
<td>15</td>
<td>INST</td>
<td>32</td>
<td>AD14/P4.6</td>
<td>49</td>
<td>HSO1</td>
<td>66</td>
<td>ANGND</td>
</tr>
<tr>
<td>16</td>
<td>ALE</td>
<td>33</td>
<td>AD15/P4.7</td>
<td>50</td>
<td>HSO0</td>
<td>67</td>
<td>ACH4/P0.4</td>
</tr>
<tr>
<td>17</td>
<td>RD</td>
<td>34</td>
<td>T2CLK/P2.3</td>
<td>51</td>
<td>HS13/HSO5</td>
<td>68</td>
<td>ACH5/P0.5</td>
</tr>
</tbody>
</table>

* Top View Looking Through Package
## DOMESTIC SALES OFFICES

<table>
<thead>
<tr>
<th>State</th>
<th>Address Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALABAMA</td>
<td>Intel Corp, 5015 Bradford Drive Suite 2, Huntsville 35805 Tel (205) 830-4070</td>
</tr>
<tr>
<td>ARIZONA</td>
<td>Intel Corp, 11255 N 28th Drive Suite 2, Phoenix 85029 Tel (602) 888-4980</td>
</tr>
<tr>
<td>CALIFORNIA</td>
<td>Intel Corp, 100 S. 20th Street Suite 116, San Jose 95110 Tel (408) 744-8500</td>
</tr>
<tr>
<td>CONNCTICUT</td>
<td>Intel Corp, 605 E. Main Street Suite 4, Hartford 06103 Tel (203) 545-0600</td>
</tr>
<tr>
<td>FLORIDA (Cont'd)</td>
<td>Intel Corp, 11200 4th Street South South St. Petersburg 33702 Tel (813) 577-5743</td>
</tr>
<tr>
<td>GEORGIA</td>
<td>Intel Corp, 3300 Denver Parkway Suite 200 Norcross 30092 Tel (404) 449-0541</td>
</tr>
<tr>
<td>ILLINOIS</td>
<td>Intel Corp, 2500 Gulf Road Suite 26 Rolling Meadows 60008 Tel (224) 861-1600 TWX 930-951-5881</td>
</tr>
<tr>
<td>IOWA</td>
<td>Intel Corp, 8777 Purdue Road Suite 125 Indianapolis 46208 Tel (317) 875-2603</td>
</tr>
<tr>
<td>KANSAS</td>
<td>Intel Corp, 1930 S Andrews Drive N.E. City Rapids 53010 Tel (309) 385-5350</td>
</tr>
<tr>
<td>LOUISIANA</td>
<td>Intel Corp, 8400 N 11th Street Overland Park 66210 Tel (913) 842-6200</td>
</tr>
<tr>
<td>MAINE</td>
<td>Intel Corp, 7321 Parkway Drive South Suite G1075 Tel (704) 795-7050 TWX 720-892-1644</td>
</tr>
<tr>
<td>MARYLAND</td>
<td>Intel Corp, 7833 Walter Drive Greenbelt 20770 Tel (301) 441-1020</td>
</tr>
<tr>
<td>MASSACHUSETTS</td>
<td>Intel Corp, 27 Industrial Avenue Chicopee 01021 Tel (413) 258-1900 TWX 510-343-0633</td>
</tr>
<tr>
<td>MICHIGAN</td>
<td>Intel Corp, 71701 Orchard Lake Road Suite 100 West Bloomfield 48323 Tel (313) 851-9096</td>
</tr>
<tr>
<td>MINNESOTA</td>
<td>Intel Corp, 3500 W 8th Street Suite 100 Bloomington 55431 Tel (651) 335-6124 TWX 950-576-2667</td>
</tr>
<tr>
<td>MISSOURI</td>
<td>Intel Corp, 4209 Eastern Expressway Suite 123 East City 63045 Tel (614) 295-1900</td>
</tr>
<tr>
<td>NEW JERSEY</td>
<td>Intel Corp, 8500 Merrill Boulevard N.E. Suite B-295 Albuquerque 87112 Tel (505) 982-8506</td>
</tr>
<tr>
<td>NEW MEXICO</td>
<td>Intel Corp, 300 Vanderbilt Motor Parkway Hauppauge 11788 Tel (631) 255-2900 TWX. 510-227-6236</td>
</tr>
<tr>
<td>NEW YORK</td>
<td>Intel Corp, 211 White Canoe Boulevard Suite 105 Greenbelt 20770 Tel (703) 443-3510</td>
</tr>
<tr>
<td>NORTH CAROLINA</td>
<td>Intel Corp, 2700 Landmark Road Suit 102 Raleigh 27607 Tel (919) 791-0622</td>
</tr>
<tr>
<td>OHIO</td>
<td>Intel Corp, 6500 Poe Avenue Dayton 45414 Tel (513) 900-2900 TWX 810-452-2528</td>
</tr>
<tr>
<td>PENNSYLVANIA</td>
<td>Intel Corp, 450 N. Sunnyvale Road Suite 120 Channahon 60410 Tel (630) 404-9041</td>
</tr>
<tr>
<td>WASHINGT N</td>
<td>Intel Corp, 11200 4th Street South Suite 100 St. Petersburg 33702 Tel (727) 332-0100</td>
</tr>
<tr>
<td>WISCONSIN</td>
<td>Intel Corp, 450 N. Sunnyvale Road Suite 120 Channahon 60410 Tel (630) 404-9041</td>
</tr>
</tbody>
</table>

*Field Application Location*
# DOMESTIC DISTRIBUTORS

<table>
<thead>
<tr>
<th>STATE</th>
<th>CITY</th>
<th>ADDRESS</th>
<th>PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW JERSEY (Cont'd)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pioneer NorthEast Electronics</td>
<td>45 Route 46</td>
<td>Franklin Lakes 07417</td>
<td>(201) 977-3754</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fairfield 07006</td>
<td>(201) 321-6632</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW MEXICO</td>
<td>Albuquerque</td>
<td>8115 SE</td>
<td>(505) 258-6500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(505) 258-6502</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW YORK</td>
<td></td>
<td>383 Route 46 W</td>
<td>(201) 874-3000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(201) 874-3000</td>
</tr>
<tr>
<td>Arrow Electronics, Inc</td>
<td>25 Hub Drive</td>
<td>Melville 11747</td>
<td>(516) 433-0610</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(516) 433-0631</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OHIO</td>
<td>Dayton</td>
<td>45429</td>
<td>(513) 433-0610</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(513) 433-0631</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PENNSYLVANIA (Cont'd)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pioneer Electronics</td>
<td>261 Gibraltar Rd</td>
<td>Norwalk 06201</td>
<td>(203) 647-4000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(203) 647-4000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WISCONSIN</td>
<td></td>
<td>430 W. Roosevelt Avenue</td>
<td>(608) 834-9990</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(608) 834-9990</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CANADA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alberta</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>British Columbia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minnesota</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manitoba</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ontario</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quebec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EUROPEAN SALES OFFICES

BELGIUM
Intel Corporation SA
Parc Senal
Rue du Moulin a Papier 51
B-1980 Brussels
Tel: (02) 565 07 11
TELEX: 24814

DENMARK
Intel Denmark A/S
Gammelgade 81 - 3rd Floor
DK-2100 Copenhagen Ø
Tel: (35) 19 80 55
TELEX: 19567

FINLAND
Intel Finland OY
Haminantie 135
SF-00210 Helsinki
Tel: (0) 976 08 85
TELEX: 123 332

FRANCE
Intel Corporation, S.A.R.L
5 Rue de la Balance
94569 Rungis Cedex
Tel: (1) 30 24 00 21
TELEX: 270476

FRANCE (Cedex)
Intel Corporation, S.A.R.L
Immeuble BBL
14 Rue des Etoiles
92200 Neuilly-sur-Seine
Tel: (1) 46 40 89
TELEX: 303163

WEST GERMANY
Intel Semiconductor GmbH
Toidenstrasse 27
D-4000 Dusseldorf 2
Tel: (0211) 62 50 00
TELEX: 62 2377 INTL D

Intel Semiconductor GmbH
Marvinstrasse 70
D-8800 Stuttgart 1
Tel: (0711) 75 06 74
TELEX: 049 4818 INTW D

Intel Semiconductor GmbH
Breubergstrasse 61
D-73112 Feuchtwangen
Tel: (0 71 58) 00 82
TELEX: 2724628 INT D

Israël
Intel Semiconductor Ltd.*
P.O. Box 1659
Haifa 31000
Tel: (0 3) 615 28 01
TELEX: 460191

ITALY
Intel Corporation Italia Spa*
Mialetti, Piazza E 2004 (Milano)
Tel: (02) 824 00 06
TELEX: 315682 INTML

NETHERLANDS
Intel Semiconductor Nederland B.V.*
Alexanderpolder Building
Martin Meesweg 93
3068 Nijmegen
Tel: (0 40) 21 23 77
TELEX: 22269

NORWAY
Intel Norway A/S
P.O. Box 92
Oslo 3
Tel: (02) 47 00 42
TELEX: 18078

SPAIN
Intel Ibérica
Calle Zurbano 28
Madrid 04
Tel: (91) 414 40 04
TELEX: 40880

SWEDEN
Intel Sweden A.B.*
Dagvagen 24
S-17156 Solna
Tel: (08) 549 01 00
TELEX: 123261

SWITZERLAND
Intel Semiconductor A.G.*
Talackerstrasse 17
ETH, Rüti-Zürich
CH-8093 Zürich
Tel: (01) 323 29 77
TELEX: 57996 IOCH CH

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.*
Pipers Way
Oxford, Oxfordshire
OX4 3TP
Tel: (0990) 888 326
TELEX: 644440 INT SW

*Field Application Location

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA
Bacher Elektronische Geräte GmbH
Rudolfstädter Straße 26
8032 Graz
Tel: (0316) 58 46 48
TELEX: 1522 BASA A

BELGIUM
Indelo Belgium S.A.
Avenue des Croix de Guerre 54
B-1040 Brussels
Tel: (02) 34 16 34
TELEX: 254411

DENMARK
ITT Multicomponent A/S
Næstved 20
DK-2800 Gormstrup
Tel: (033) 43 86 45
TX: 33535

FINLAND
Oy Fintronik AB
Maahouitatu 24 A
SF-00760 Helsinki
Tel: (0) 299 60 24
TELEX: 124 244 Finon SF

FRANCE
General
2 J. de Courtabouf
Avenue de la Faciole
92300 Levallois-Perret
Tel: (1) 307 78 78
FAX: 265 1700

GERMANY
Kron & Hartman
Kron & Hartmann AG
Engelbrechtstrasse 12
6900 Mannheim
Tel: (0 62 24) 00 50
TELEX: 62590 EDC D

Jernm 11 GmbH
Apostelstrasse 2
6227 Bad Camberg
Tel: (061 50 1) 49 64
TELEX: 94492 JERN D

CES Computer electronics Systems GmbH
Gutenbergstrasse 4
2384 Hanseatic-Lübeck
Tel: (0410 2) 4204
TELEX: 21095260

SCHNELL GmbH
Hansinenstrasse 15
8000 Munich 21
Tel: (089) 57 20 84
TELEX: 91 728189

PRODUCTION Vertriebs GmbH
Max Planckstrasse 1-3
6072 Dreischei bei Frankfurt
Tel: (069) 230564
TELEX: 4797893

IRELAND
Micro Marketing
12 Lower Ormond Street
Dublin 2
Tel: (0) 35 62 88
TELEX: 257898

ISRAEL
Electronics Ltd.
11 Rotshild Street
10600 Tel Aviv
Tel: (0) 67 51 64
TELEX: 326838

ITALY
Unicroni S.P.A.
Viale Europa, 18
20154 Milano
Tel: (0 2) 34 67 51
TELEX: 332332

TELECOM
Mediterraneo Pol E/F
30000 Asseco
Tel: (023) 624 6480

TELECOM
Valeggio del Mezzogiorno
1 Box 180
958 930 Bologna
Tel: (0 09 08 08 20
TELEX: 11941

NETHERLANDS
Koning & Hartman
Enghbrechtstrasse 12
6900 Mannheim
Tel: (0 62 24) 00 50
TELEX: 62590 EDC D

NORWAY
Nordisk Elektronik (Norge) A/S
Postoffice Box 123
0034 Oslo 4
Tel: (02) 298 210
TELEX: 17546

PORTUGAL
Dutra Componentes E Electronica LDA
Miguel Bombarda, 133
4000 Porto
Tel: (2) 249 1821
TELEX: 141 1482 BINES-P

SPAIN
Sastre S.A.
Av. Pomplou, Fabra 12
28035 Madrid
Tel: (0 1) 219 50 11
TELEX: 78508 ATE D

ITT SESA
Miguel Angel 21, 6 Piso
Madrid 10
Tel: (044) 18544 00
TELEX: 274641

SWEDEN
AB Bofors Beckenham
Sallingsgatan 22
10201 Stockholm
TELEX: 75050 ATE D

Nordisk Elektronik AB
Box 27301
Sandvagnsgatan 71
10254 Stockholm
Tel: (0 835) 564 00
TELEX: 105672

Teko AB
Vilhelmsdalen 1
Box 186
S-181 26 Bromma
Tel: (0 08) 98 08 20
TELEX: 11941

SWITZERLAND
Industrialstrasse 21
CH-8800 Winterthur
Tel: (0 61) 623 20 40
TELEX: 59766 NEDEL CH

UNITED KINGDOM
Byler Ltd
Luti 57
London, England
Tel: (0) 734 85 30 40
TELEX: 649210

Conway Microsystems Ltd
44 The Square
St Albans, Hertfordshire
TELEX: 4014414

CMB Ltd
M.F.D.
East Lane Road
North Wembley
Middlesex, HA9 7PQ
Tel: (0190) 493077

JEANet Industries
Verny Manor
Servidorix Village,
Kent
Tel: (0272) 345678

M.F.D.
East Lane Road
North Wembley
Middlesex, HA9 7PQ
Tel: (0190) 493077

Rapid Recall, Ltd
Pend-Handa/Denmark St
High Wycombe
Bucks, England
Tel: 2352 GEL 26 271

U.S.S.R.
587047

YUGOSLAVIA
H. R. Microelectronics Enterprises
P.O. Box 35600
San Josè, California 95050
Tel: (408) 2879000
TELEX: 278-559
# INTERNATIONAL SALES OFFICES

**AUSTRALIA**
Intel Australia Pty Ltd.
P.O. Box 571
North Sydney NSW, 2065
Tel 011-61-2-957-2744
Fax 011-61-2-957-2744

**JAPAN**
Intel Japan K.K.
5-4 Tokodai, Toyospa-machi
Tsurumai, Nagoya, 464-8526
Tel 052974-6501
Fax 052974-6502

**JAPAN (Comt'n)**
Intel Japan K.K.
1-5-1 Minowa
Chuo-ku, Tokyo 105
Tel 03-585-8511
Fax 03-585-8512

**JAPAN (Comt'n)**
Intel Japan K.K.
1-23-3 Shinagawa
Shinagawa-ku, Tokyo 154
Tel 03-542-2424
Fax 03-542-2425

**KOREA**
Intel Semiconductor Asia Ltd.
Seongbuk Blvd., N.H. Floor #906
Seoul 150
Tel 011-82-2-788-8196 or 8286
Fax 011-82-2-788-8191

**INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES**

**ARGENTINA**
VLC S.R.L.
Santamaria 1630, 1 Piso
5042 Buenos Aires
Tel 011-54-1-35-1201/9242
Fax 011-54-1-35-1201

**BRAZIL**
Isoptron S.A.
0515 Av. Martins 3650-6 Andar
Pacifica, Pacheco Res.
Sao Paulo, 02154-365
Tel 011-55-11-833-2572
Fax 011-55-11-833-2573

**CHILE**
DIN
Arcadia
P.O. Box 8522
Santiago, 8759
Tel 011-56-2-277-564
Fax 011-56-2-277-564

**JAPAN**
Auptech Electronics Co., Ltd.
Kawagoe Bldg. 407
1-1-1 Aoyama
Minato-ku, Tokyo 107
Tel 03-585-8511
Fax 03-585-8512

**KOREA**
Samsung Electronics Co., Ltd.
Seongbuk Blvd., NH Floor #906
Seoul 150
Tel 011-82-2-788-8196 or 8286
Fax 011-82-2-788-8191

**INTERNATIONAL SALES OFFICES**

**HONG KONG**
Intel Semiconductor Ltd.
175-3 Queen's Centre
25/F, 1 Queen's Road West
Tel 011-852-5-212-331
Fax 011-852-5-212-331

**INDIA**
Micron Devices 60, Arnul Complex
D. V. G. Road
Bangalore 560004
Fax 011-81-906-600-631

**KOREA**
Intel South Korea
Korea Telecom Centre
25-3, Nongdong-dong
Youngdong-gu, Seoul 110
Tel 031-589-8500
Fax 031-589-8501

**PHILIPPINES**
Lantronics, Philcoast Corp.
1111 E. Rodriguez, Jr. Avenue
Valencia Polo, Quezon City
Tel 011-63-2-587-0377
Fax 011-63-2-587-0377

**SOUTH AFRICA**
Electronic Building, Elements, P.O. Box 4006
Pretoria
Tel 011-501-2901
Fax 011-501-2901

**TURKEY**
Intel Semiconductor Ltd.
175-3 Queen's Centre
25/F, 1 Queen's Road West
Tel 011-852-5-212-331
Fax 011-852-5-212-331

**U.S.A.**
Intel Corporation
4000 E. Pacific Highway
San Jose, California 95138
Fax 408-748-5747

**INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES**

**ARGENTINA**
VLC S.R.L.
Santamaria 1630, 1 Piso
5042 Buenos Aires
Tel 011-54-1-35-1201/9242
Fax 011-54-1-35-1201

**BRAZIL**
Isoptron S.A.
0515 Av. Martins 3650-6 Andar
Pacifica, Pacheco Res.
Sao Paulo, 02154-365
Tel 011-55-11-833-2572
Fax 011-55-11-833-2573

**CHILE**
DIN
Arcadia
P.O. Box 8522
Santiago, 8759
Tel 011-56-2-277-564
Fax 011-56-2-277-564

**JAPAN**
Auptech Electronics Co., Ltd.
Kawagoe Bldg. 407
1-1-1 Aoyama
Minato-ku, Tokyo 107
Tel 03-585-8511
Fax 03-585-8512

**KOREA**
Samsung Electronics Co., Ltd.
Seongbuk Blvd., NH Floor #906
Seoul 150
Tel 011-82-2-788-8196 or 8286
Fax 011-82-2-788-8191

**PHILIPPINES**
Lantronics, Philcoast Corp.
1111 E. Rodriguez, Jr. Avenue
Valencia Polo, Quezon City
Tel 011-63-2-587-0377
Fax 011-63-2-587-0377

**SOUTH AFRICA**
Electronic Building, Elements, P.O. Box 4006
Pretoria
Tel 011-501-2901
Fax 011-501-2901

**TURKEY**
Intel Semiconductor Ltd.
175-3 Queen's Centre
25/F, 1 Queen's Road West
Tel 011-852-5-212-331
Fax 011-852-5-212-331

**U.S.A.**
Intel Corporation
4000 E. Pacific Highway
San Jose, California 95138
Fax 408-748-5747
DOMESTIC SERVICE OFFICES

CALIFORNIA
Intel Corp
1330 Shoreline Way
Mt. View 94030
Tel: (415) 335-3731
FAX: 800-335-9279
TDD: 336-0238
Intel Corp
2000 E. 4th Street
Suite 110
San Diego 92121
Tel: (714) 561-2475
Intel Corp
4350 Executive Drive
Suite 150
San Diego 92121
Tel: (888) 452-9800
Intel Corp
5559 N. Canyon Avenue
Suite 120
Torrance 90505
Tel: (213) 708-0333
COLORADO
Intel Corp
650 South Cherry
Suite 720
Denver 80222
Tel: (303) 331-6286
FAX: 800-921-2299
CONNECTICUT
Intel Corp
21 Mill Plain Road
Darien, 06820
Tel: (203) 748-3120
FLORIDA
Intel Corp
1500 N.W. 62nd Street
Suite 700
P. Leander 33305
Tel: (305) 772-8020
FAX: 513-958-9407
FLORIDA (Cont'd)
Intel Corp
600 N. Mallard Avenue
Suite 205
Seeking 33323
Tel: (305) 438-2393
FAX: 800-853-9219
GEORGIA
Intel Corp
2300 South Parkway
Suite 320
Atlanta 30339
Tel: (404) 441-1711
ILLINOIS
Intel Corp
1250 Golf Road
Suite 300
Schaumburg 60173
Tel: (847) 981-7098
FAX: 800-921-1713
KANSAS
Intel Corp
4400 W. 110th Street
Overland Park 66210
Tel: (913) 440-6000
MARYLAND
Intel Corp
5th Floor Product Service
2633 Walker Drive
Baltimore 21209
Tel: (301) 441-0920
MASSACHUSETTS
Intel Corp
27 Industrial Avenue
Chelmsford 01824
Tel: (978) 258-8000
FAX: 800-943-0333
MICHIGAN
Intel Corp
7071 Orchard Lake Road
Suite 100
Southfield 48033
Tel: (313) 851-8505
MINNESOTA
Intel Corp
2203 East City Expressway
Suite 143
Minneapolis 55401
Tel: (763) 391-2010
NEW JERSEY
Intel Corp
385 Silver Avenue
Englewood Cliffs 07632
Tel: (201) 440-2000
FAX: 800-921-1993
NEW YORK
Intel Corp
Harlem Plaza III
Harlem Center
Eldon 06817
Tel: (201) 225-3000
NORTH CAROLINA
Intel Corp
2300 W. Meadowview Road
Suite 205
Greensboro 27407
Tel: (919) 234-7541
OHIO
Intel Corp
Chagrin/Beachwood Bldg
Suite 100
28000 Chagrin Boulevard
Cleveland 44122
Tel: (216) 484-8019
FAX: 800-427-8295
INTEL CORP
6000 Polk
Dayton 45414
Tel: (513) 890-5550
OREGON
Intel Corp
10700 S.W. Beaverton-Hilsdale Highway
Beaverton, 97005
Tel: (503) 641-4066
FAX: 800-457-8711
INTEL CORP
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-3000
PENNSYLVANIA
Intel Corp
201 Penn Center Boulevard
Suite 300 W
Pittsburgh 15235
Tel: (412) 354-1540
TEXAS
Intel Corp
333 E. Anderson Lane
Suite 614
Austin 78752
Tel: (512) 454-3628
FAX: 512-874-1247
Intel Corp
12300 Ford Road
Suite 200
Dallas 75204
Tel: (214) 341-8087
FAX: 915-860-5817
WASHINGTON
Intel Corp
110 11th Avenue N.E.
Suite 310
Seattle 98109
Tel: 1-800-425-3500
FAX: 910-443-3002
WISCONSIN
Intel Corp
450 N. Sunnyvale Road
Suite 120
Brookfield 53005
Tel: (414) 784-9087