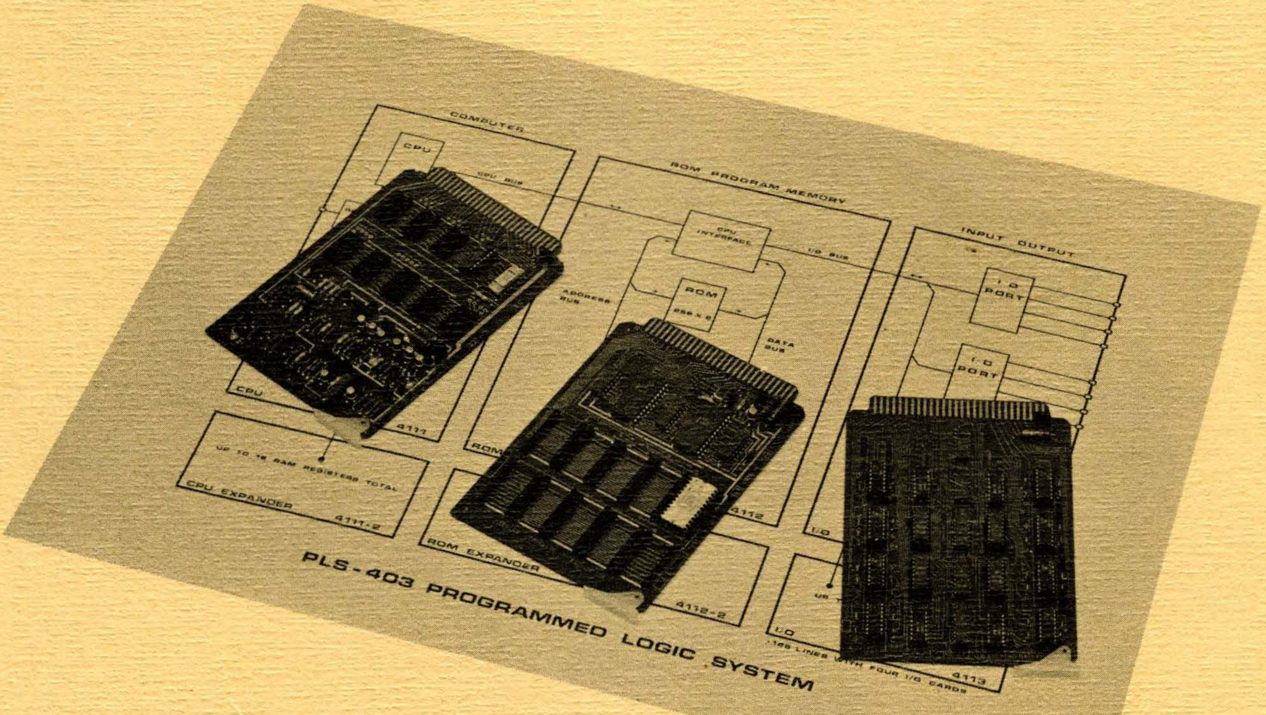
THE DESIGNERS GUIDE TO PROGRAMMED LOGIC

For PLS 400 Systems



PRO-LOG CORPORATION
852 Airport Road
Monterey, California

THE
DESIGNERS GUIDE
TO PROGRAMMED LOGIC

For PLS 400 Systems

Written by

Matt Biewer

PRO-LOG CORPORATION

852 Airport Road

Monterey, California

Price: $10.00

15 August 1973

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Concluded)

LIST OF TABLES

LIST OF ILLUSTRATIONS

1. SUMMARY

PLS 400 systems are micro-programmable processing systems suitable for implementing random logic, and numeric and small alpha numeric data handling applications. As shown in Figure 1-1, the PLS-400 system consists of a CPU, RAM register storage, ROM program memory, and input/output ports. The CPU is an Intel 4004 CPU chip, RAM registers are the Intel 4002 circuits, ROM program memory is implemented using Intel 1702 or NSC 5202 erasable reprogrammable ROM's or equivalents, and I/O ports are TTL latches and selectors.

CPU CAPABILITIES

- Twelve bit program address
- Three level address stack for subroutines
- Sixteen, four bit index registers
- A four bit accumulator plus carry
- One, eight bit instruction word per cycle
- Forty one single word instructions
- Five double word instructions
- Arithmetic and accumulator instructions

  Add and subtract with carry

  Complement, rotate, increment, decrement, clear, and load the accumulator

  Complement, clear, and set carry
- Decimal arithmetic instructions
- Decision making (address control instructions)

  Test accumulator for zero or nonzero

  Test carry for logic one or zero

  Test external input for high or low

  Increment any index register and test for zero
- Input/output instructions
- RAM register instructions
- Register instructions

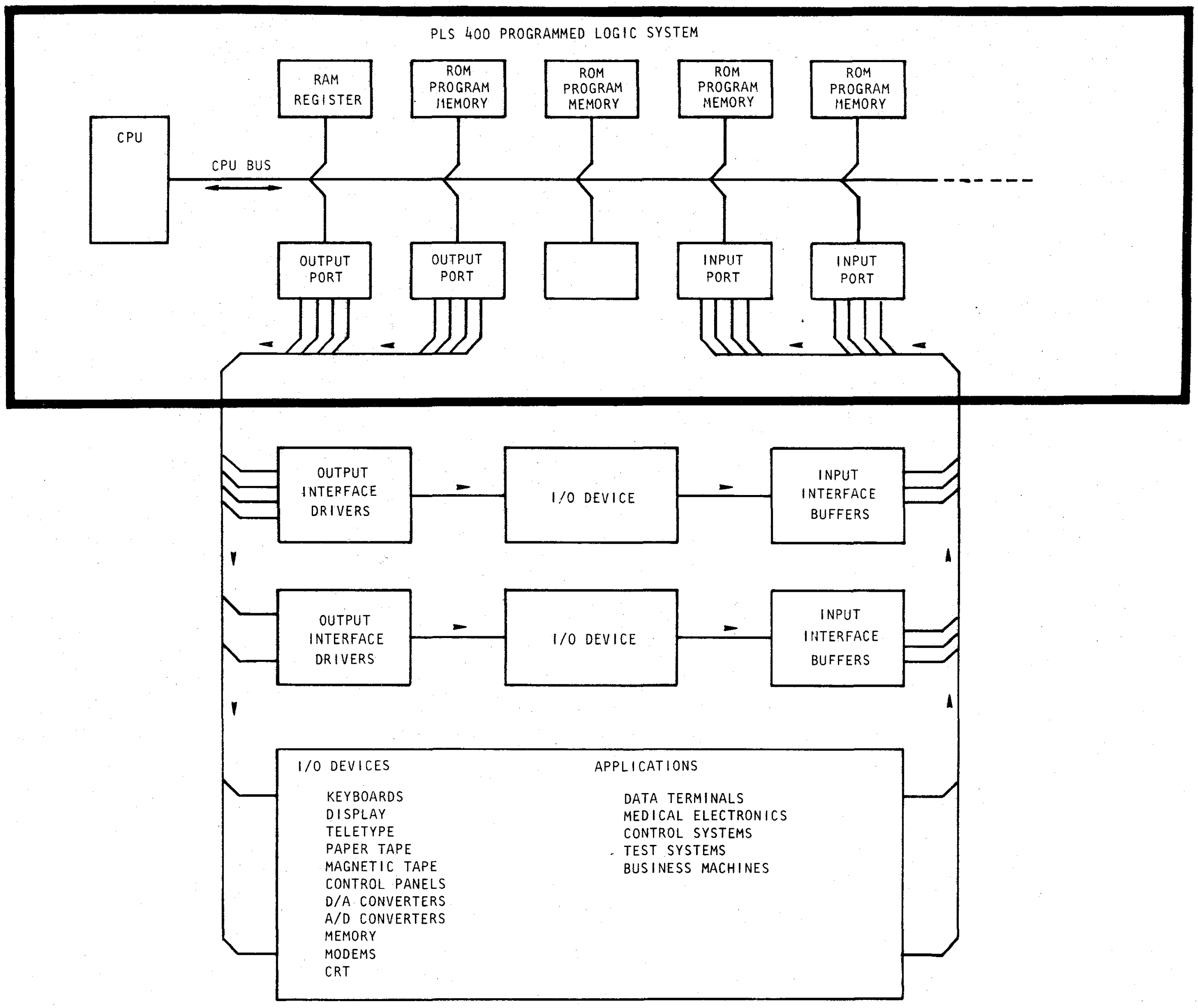


FIGURE 1-1

PLS-400 Programmed Logic System Application

ROM PROGRAM MEMORY

- Programmable erasable ROM's MSC 5202, Intel 1702
- 256 eight bit instructions words per page
- Sixteen pages maximum (ROM's) 4096 words of instruction

RAM REGISTER STORAGE

- Intel 4002 RAM
- Organized as four registers of 16 four bit words plus four status words for each register
- Requires CPU instruction addressing
- Includes four output lines used with CPU output instruction

INPUT/OUTPUT

- TTL output latches and input selectors
- Requires CPU selection by instruction
- 128 lines directly selectable
- Input instruction, gates data into the CPU accumulator
- Output instruction, latches accumulator data at output

2. PLS 400 HARDWARE

The PLS 400 series provides a choice of micro-processor card sets with varying expandability. Each set provides the CPU, ROM program memory, RAM register storage and I/O. All sets are implemented with CPU clock and external reset and power-on reset circuits. ROM program memory on each card set is implemented with programmable erasable ROM. The use of erasable, reprogrammable ROMs provides a speedy tool for implementing programmed logic. The PLS 401 one card set is complete on a single card providing the lowest cost for limited system size. The system expansion is limited to 1024 words of ROM program memory, 320 characters of RAM register storage, four output ports, four input ports, and one RAM output port.

The PLS 402 two card set provides for reasonable program memory and extensive I/O. The system expansion is limited to 1536 words os program memory, 320 characters of RAM register storage, 4 RAM output ports, and up to 128 I/O lines.

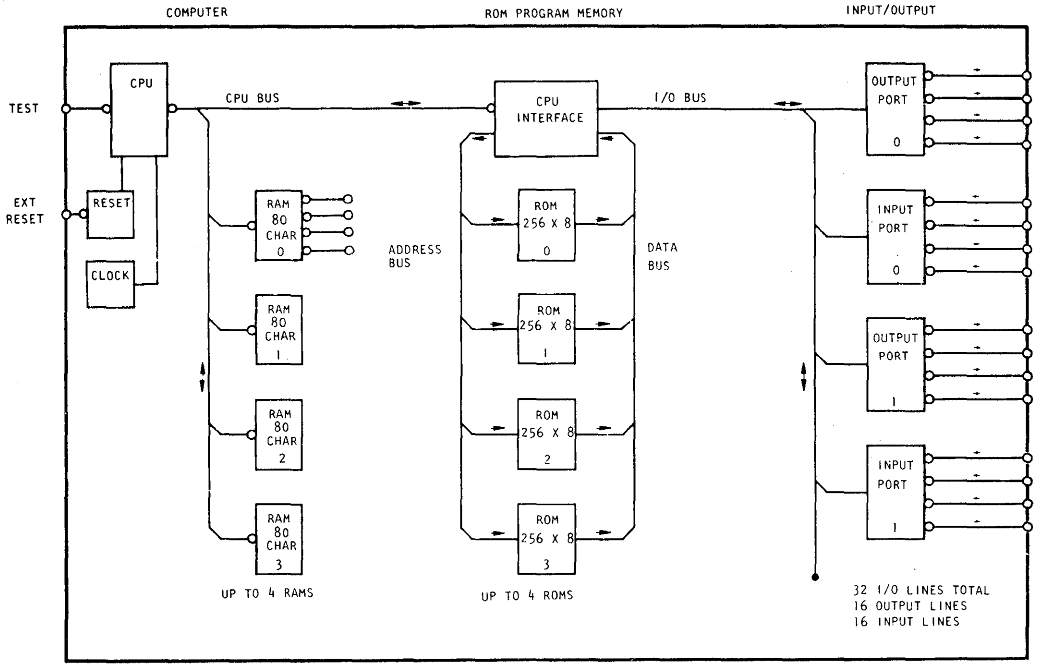
The PLS 403 three card set provides maximum expansion capability on ROM, RAM and I/O. The system expansion is unlimited to the full CPU capability of 4096 words of program memory, 1280 characters of RAM register storage, 48 lines of RAM output, and 128 lines of I/O. The PLS 403 CPU Card 4111 will accept either eight 4002 RAM register devices or eight 4001 masked ROM devices. This card in itself can become a one card system with masked ROMs.

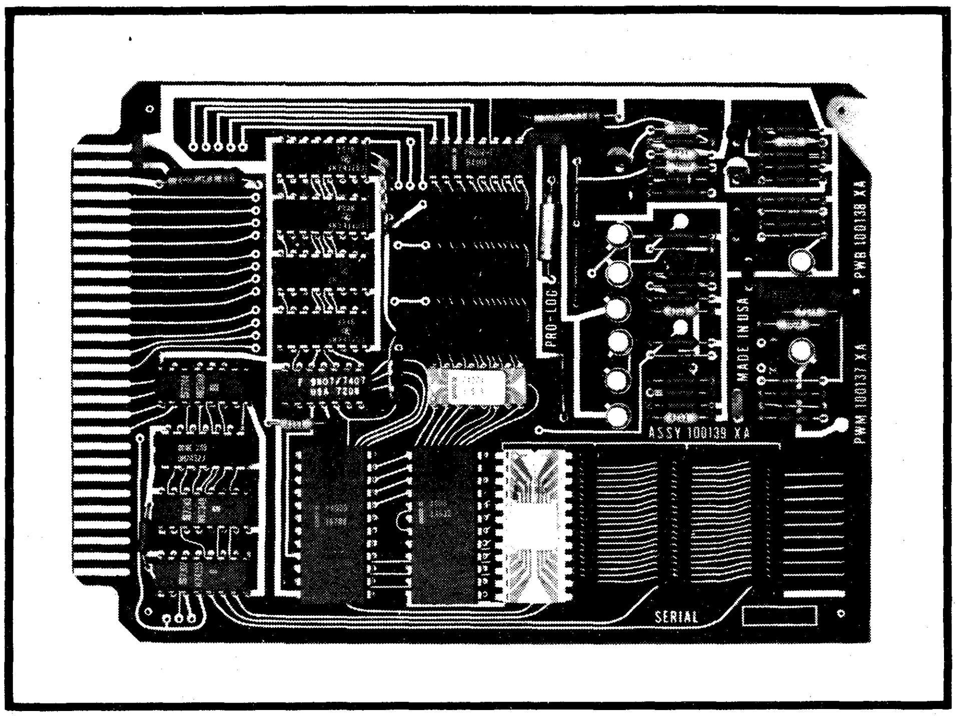
PLS-401 SINGLE CARD SYSTEM

A programmable logic system which implements the Intel MCS^{TM}-4 Micro Computer Set into a working system with CPU, ROM program memory, RAM register storage and I/O on a single card. The PLS-401 organization provides for reasonable program and I/O capacity to give the lowest cost approach to investigating PLS technology.

FEATURES

- Single card programmed logic system for protypes or production
- 1024 words of ROM program memory capacity (4 ROMs)
- 320 characters of RAM register storage capacity (4 RAMs)
- Four output ports (16 lines)
- Four input ports (16 lines)
- One RAM output port (4 lines)





PLS-401 ONE CARD PROGRAMMED LOGIC

PLS-401 SPECIFICATIONS

Card Dimensions

- 4.5 inches high
- 6.5 inches long
- 0.48 inch maximum profile thickness
- 0.062 inch printed circuit board thickness

Includes:

- Card ejector
- One 4004 CPU soldered to board
- One 4002 RAM soldered to board plus three RAM sockets
- One 1702A ROM and four ROM sockets
- Master power-on and external reset circuit
- Two phase CPU clock circuit
- Four TTL output ports (16 lines)
- Four TTL input ports (16 lines)
- One MOS output port (4 lines)
- CPU test input (MOS)

Maximum System Capabilities

- Four 4002 RAMs (320 four bit characters)
- Four 1302, 1602 or 1702 ROMs (1024 words of program memory)
- 20 output lines
    - 16 TTL port lines
    - 4 MOS RAM port lines
- 16 TTL input lines

Instruction Execution Capability

- Capable of executing all 46 of the 4004 CPU Instruction except for DCL and WPM
- 10.8 microseconds instruction execution time

Logic Levels of External Connections:

- Low level active
    - TTL Port: Standard TTL compatibility and loading
    - MOS Input: Standard TTL compatibility
    - MOS Output: Drive capability, one LPTTL or one standard TTL load with 12K pull-down to -VDD

Power Requirement

+VCC = +5 volts 5% @ 550 mA maximum fully loaded (30 mA per RAM, 35 mA per ROM)
GND = 0 volts
-VDD = -10 volts 5% @ 350 mA maximum fully loaded (30 mA per RAM, 35 mA per ROM)

Connector Requirements

56 pin, 28 position dual-readout on 0.125 centers

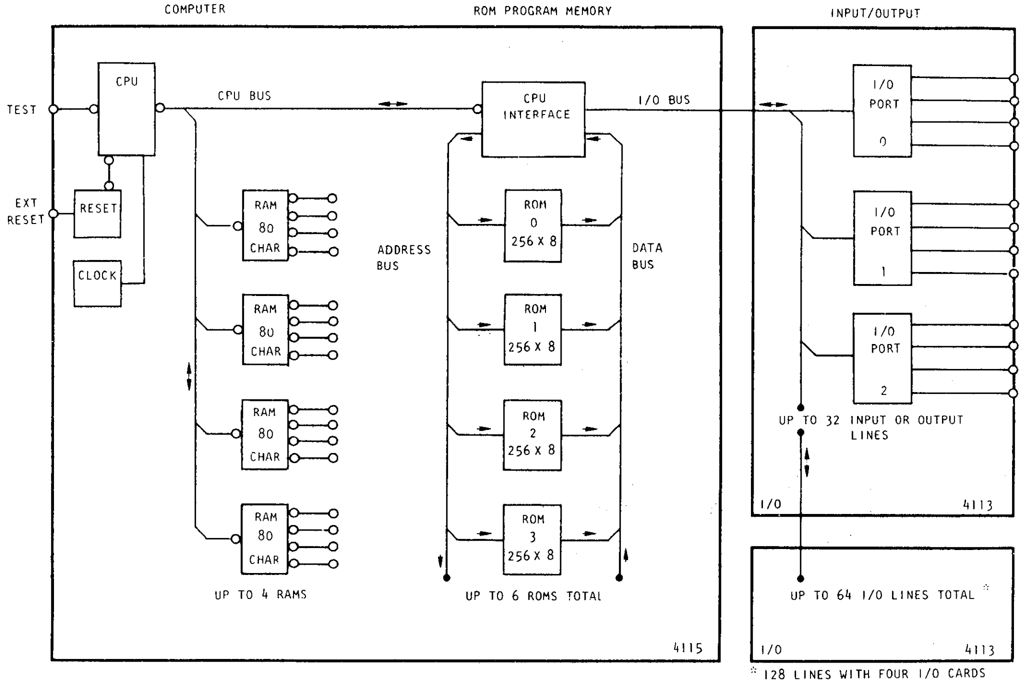
MCS^{TM} is a registered trademark
of the Intel Corporation

PLS-402 TWO CARD SYSTEM

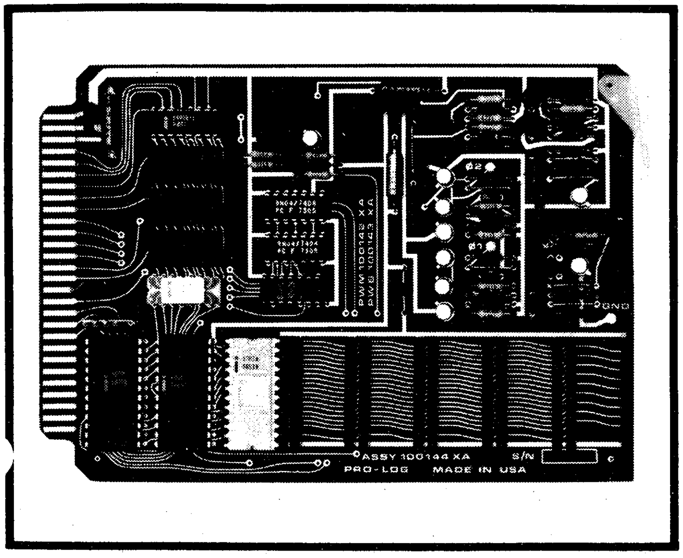
A programmable logic system which implements the Intel MCS^{TM}-4 Micro Computer Set into a working system with CPU, ROM program memory, RAM register storage and I/O on two cards. The PLS-402 organization provides for reasonalbe program capacity and unlimited I/O capacity within the MCS-4 capability. The CPU card can be applied individually or used with one or more I/O cards depending on system requirements.

FEATURES

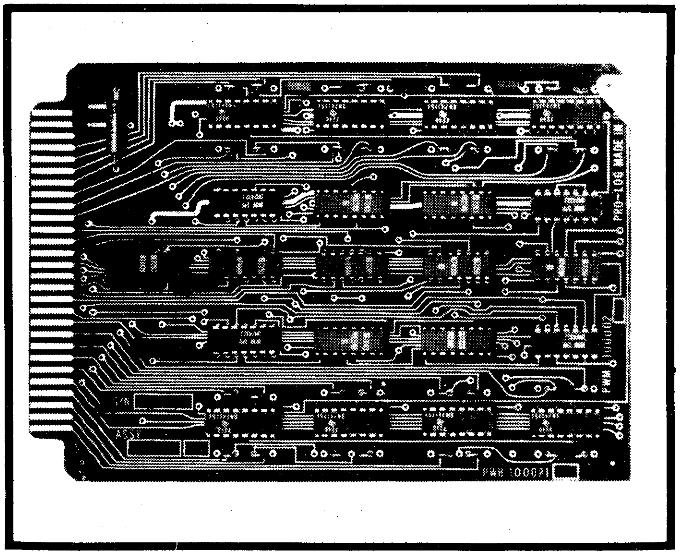
- Two card programmed logic system with expandable I/O
- 1536 words of ROM program memory capacity (6 ROMs)
- 320 characters of RAM register storage capacity (4 RAMs)
- Eight I/O ports (32 lines) expandable to 128 lines
- Four RAM output ports (16 lines)



CPU ROM



I/O



PLS-402 TWO CARD PROGRAMMED LOGIC

PLS-402 SPECIFICATIONS

Card Dimensions

4.5 inches high
6.5 inches long
0.48 inch maximum profile thickness
0.062 inch printed circuit board thickness

Includes:

One 4115 CPU card
One 4113 I/O card

CPU Card Includes:

Card ejector
One 4004 CPU soldered to board
One 4002 RAM soldered to board plus three RAM sockets
One 1702A ROM and six ROM sockets
Master power-on and external reset
Two phase CPU clock circuit
Four MOS output ports (16 lines) when four RAMs are used
One MOS CPU Test input

I/O Card Includes:

Card ejector
Eight TTL ports (32 lines)
Each port selectable as either an input port or output port
Output port lines can be wired for either high level or low level active
Common and individual reset inputs for each port

Maximum System Capabilities

Four 4002 RAMs (320 four bit characters)
Six 1302, 1602, or 1702 ROMs (1536 words of program memory)
16 MOS RAM port lines
128 TTL I/O port lines (requires four 4113 I/O cards)

64 output lines
64 input lines

Instruction Execution Capability

Executes all 46 of the 4004 CPU instructions except for DCL and WPM
10.8 microseconds instruction execution time

Logic Levels of External Connections:

Low level active

TTL Port: Standard TTL compatibility and loading
MOS Input: Standard TTL compatibility
MOS Output: Drive capability, one LPTTL or one standard TTL load with 12K pull-down to -VDD

Power Requirement: One CPU card and one I/O card both fully loaded

+VCC = +5 volts 5% @ 950 mA maximum (30 mA per RAM, 35 mA per ROM)
GND = 0 volts
-VDD = -10 volts 5% @ 450 mA maximum (30 mA per RAM, 35 mA per ROM)

Connector Requirements for each card

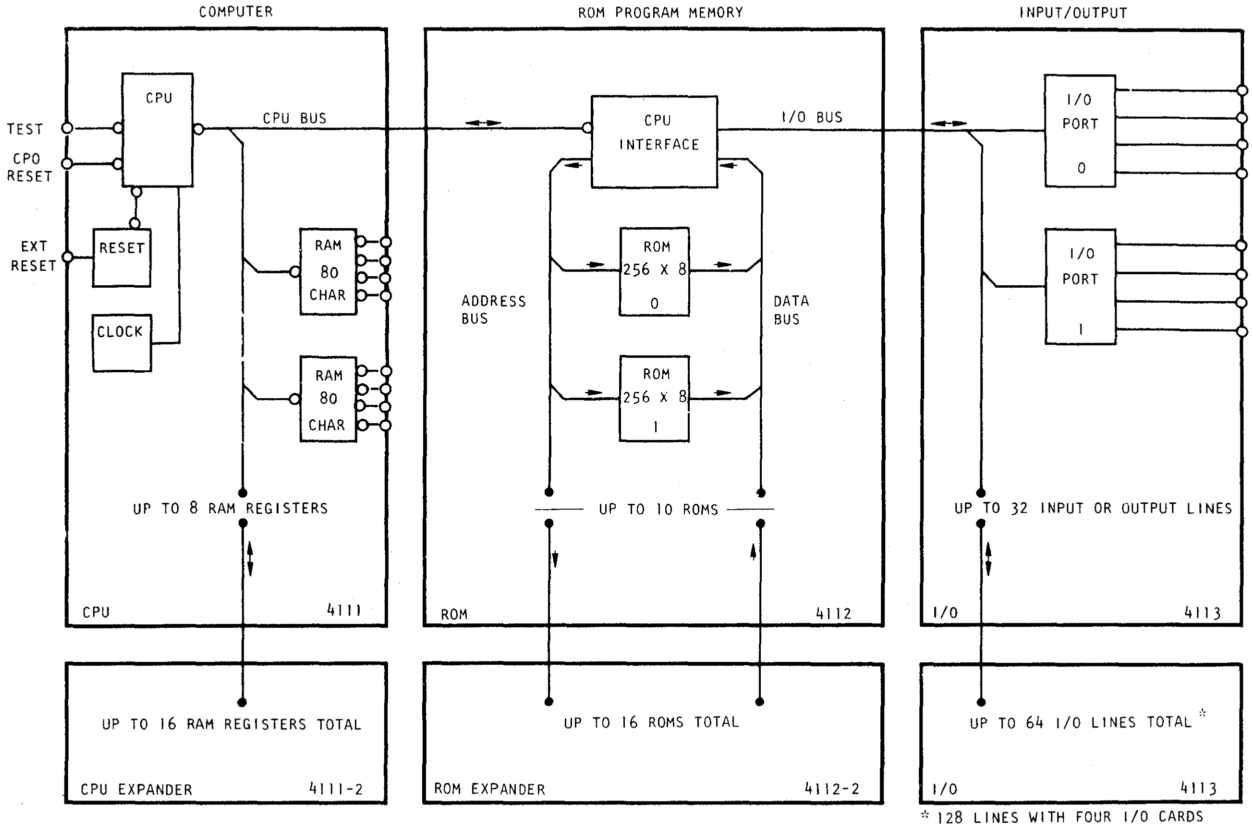
56 pin, 28 position dual-readout on 0.125 centers

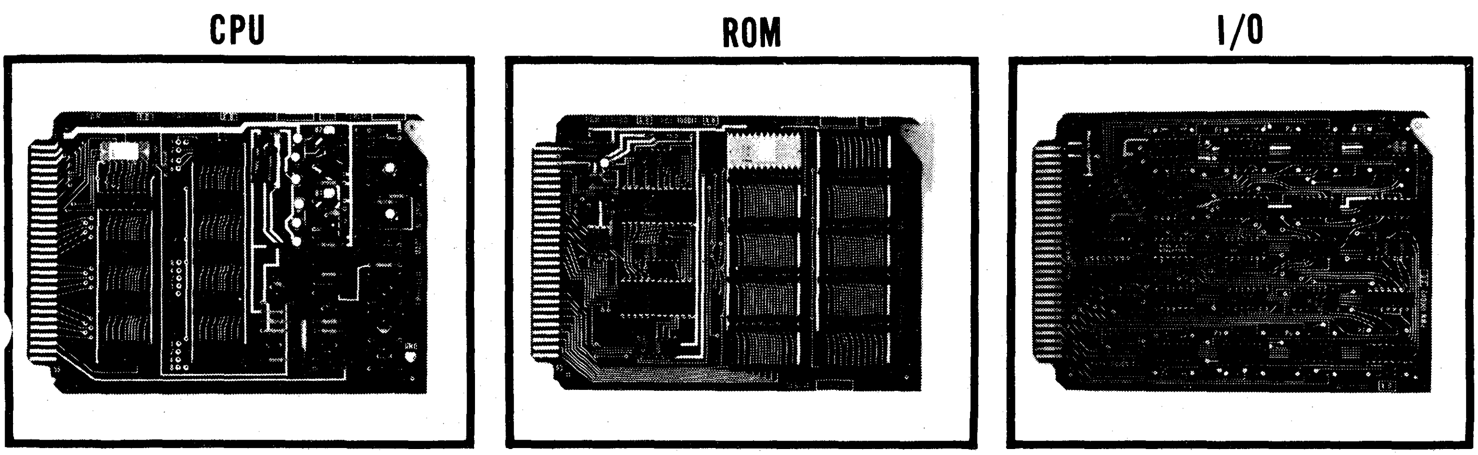
PLS-403 THREE CARD SYSTEM

A programmable logic system which implements the Intel MCS^{TM}-4 Micro Computer Set into a working system with CPU, ROM program memory, RAM register storage and I/O on three cards. The PLS-403 organization provides for unlimited program and I/O capacity within the MCS-4 capability. This modular arrangement allows the designer to tailor system size to suit his needs.

FEATURES

- Three card programmed logic system with expandable RAM, ROM and I/O
- 2560 words of ROM program memory capacity expandable to 4096 words
- 640 characters of RAM register storage capacity expandable to 1280 characters
- Eight I/O ports (32 lines) expandable to 128 lines
- Six RAM output port capacity (24 lines) expandable to 48 lines
- CPU card can be used as a single card system with masked ROMs
- Allows use of RAM program memory





CPU

ROM

I/O

PLS-403 THREE CARD PROGRAMMED LOGIC

PLS 403 SPECIFICATIONS

Card Dimensions

- 4.5 inches high
- 6.5 inches long
- 0.48 inch maximum profile thickness
- 0.062 inch printed circuit board thickness

Includes:

- One 4111 CPU card
- One 4112 ROM card
- One 4113 I/O card

CPU Card Includes:

- Card ejector
- One 4004 CPU soldered to board
- One 4002 RAM with eight RAM sockets
- Master power-on and external system reset
- Separate external CPU reset
- Two phase CPU clock circuit
- Six MOS port (24 lines) available when used with RAMs or masked ROMs
- One MOS CPU TEST input
- RAM sockets will accommodate 4001 masked ROMs

ROM Card Includes

- Card ejector
- One 1702A with 10 ROM sockets
- Signal lines for controlling RAM program memory

I/O Card Includes

- Card Ejector
- Eight TTL ports (32 lines)
- Each port is selectable as either an input port or output port
- Output port lines can be wired for either high level or low level active
- Common and individual reset inputs for each port

Maximum System Capabilities

- 16 4002 RAMs (1280 four bit characters) or 16 masked ROMs (4096 words of program memory)
- 16 1302, 1602, or 1702 ROMs (4096 words of program memory) with ROM expander 4112-2
- 12 MOS ports (RAM or 4001 masked ROM) with CPU expander 4111-2
  - 64 output lines
  - 64 input lines

Instruction Execution Capability

- Capable of executing all 46 of the 4004 CPU instructions
- 10.8 microseconds instruction execution time

Logic Levels of External Connections

Low Level active

TTL Port:	Standard TTL compatibility and loading
MOS Input:	Standard TTL compatibility
MOS Output:	Drive capability, one LPTTL or one standard TTL load with 12K pull-down to -VDD

Power Requirement: One CPU card, one ROM card, one I/O card all fully loaded

+VCC	=	+5 volts 5% @	1.3 amp maximum	(30 mA per RAM, 35 mA per ROM)
GND	=	0 volts		
-VDD	=	-10 volts 5% @	0.7 amp maximum	(30 mA per RAM, 35 mA per ROM)

Connector Requirements for each card

56 pin, 28 position dual-readout on 0.125 centers

MCS^{TM} is a registered trademark of the Intel Corporation

3. TIMING AND DEVICE DATA SHEETS

TIMING

The PLS 400 systems use a 4 bit micro-processor thus all data is communicated between the system elements in groups of 4 bits. The instruction cycle for the CPU requires eight, 4 bit time intervals as shown in Figure 3-1. The eight time intervals accomplish program memory addressing, instruction retrieval, and instruction execution. The 12 bit address, required to address up to 4096 words of program memory, is sent from the CPU to the program memory in three time intervals defined as A1, A2, and A3. The eight bits of instruction for each word are sent from the addressed program memory to the CPU in two time intervals defined as M1 and M2. During the last three time intervals defined as X1, X2, and X3 the CPU executes the instruction.

Each time interval is generated by the operation of the two phase CPU clock circuit. The two phase clock accomplishes the operations within the MOS CPU, RAM, and CPU interface devices.

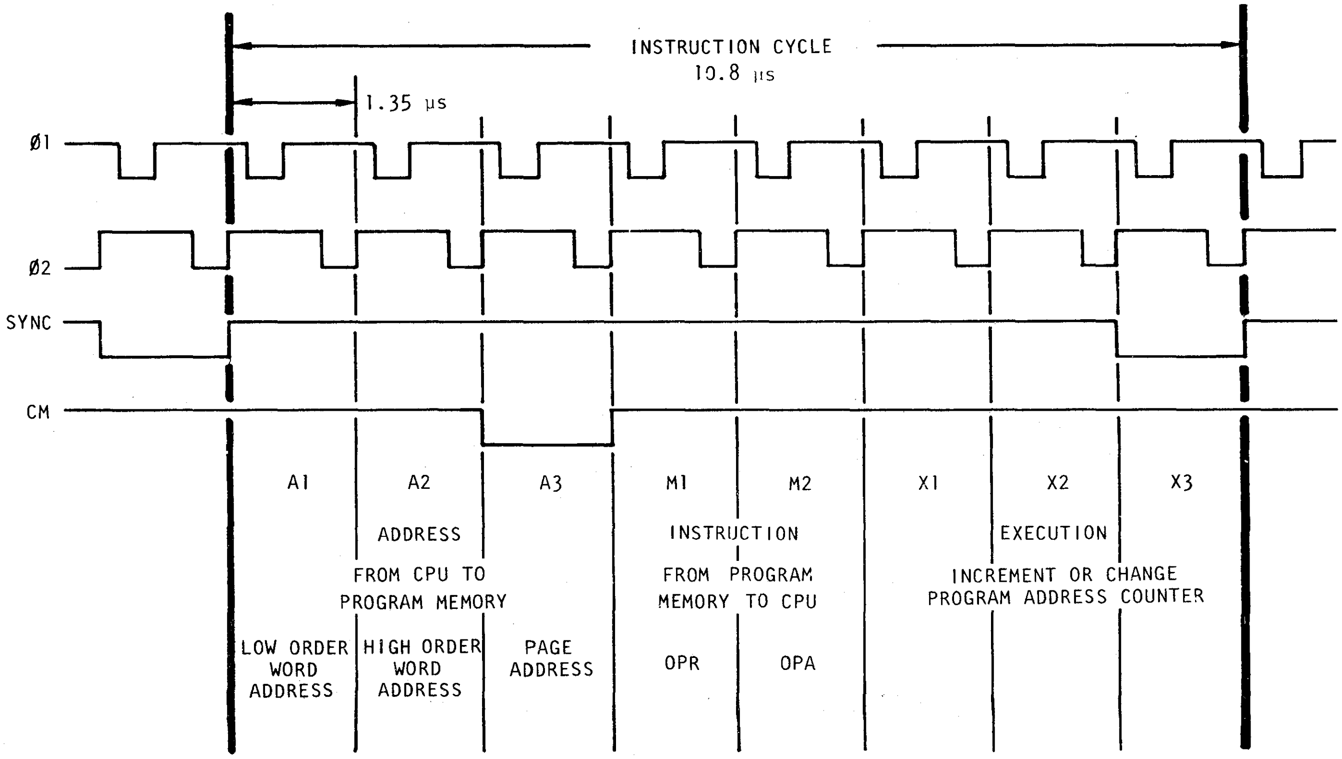


FIGURE 3-1

CPU Instruction Timing for Most Instructions

The sync pulse sent from the CPU keeps the RAM register and CPU interface devices in step with the CPU. The CM line signals the RAM registers and the CPU interface device to accept and decode chip select information on the CPU bus. CM always occurs at A3 time as this is the ROM program memory chip select (page) address. CM also occurs at X2 time as shown in Figure 3-2 during the SRC instructions for addressing RAM register devices and I/O ports, and at M2 time shown in Figure 3-3 during I/O and RAM register instructions for sending operand information to the RAM registers and the CPU interface circuits.

CM-RAM lines available from the CPU are used for bank switching of RAM register devices. If four or less RAM register devices are used on a system they may be tied to the CM line. When the CM-RAM lines are used and selected using the DCL instruction the timing is identical to the CM timing shown in Figures 3-1, 3-2, and 3-3.

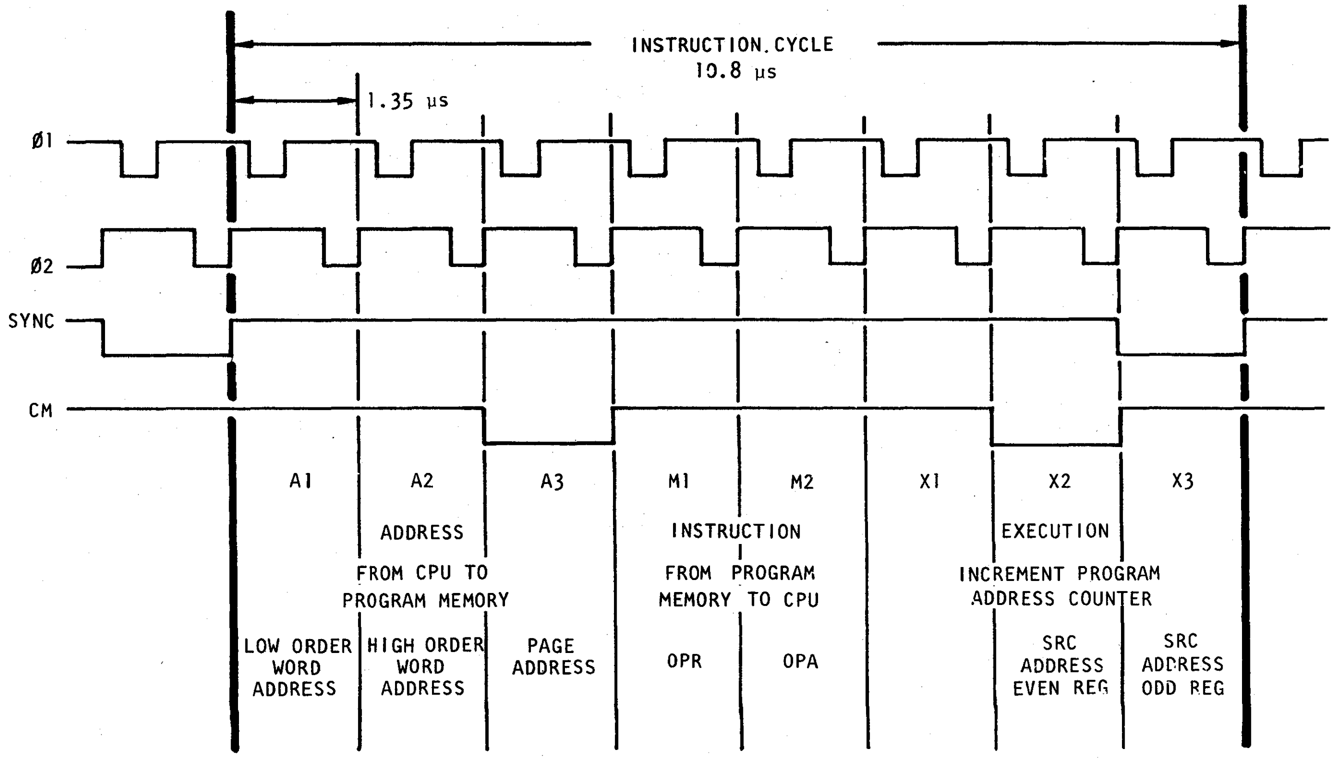


FIGURE 3-2

CPU Timing for SRC Instruction

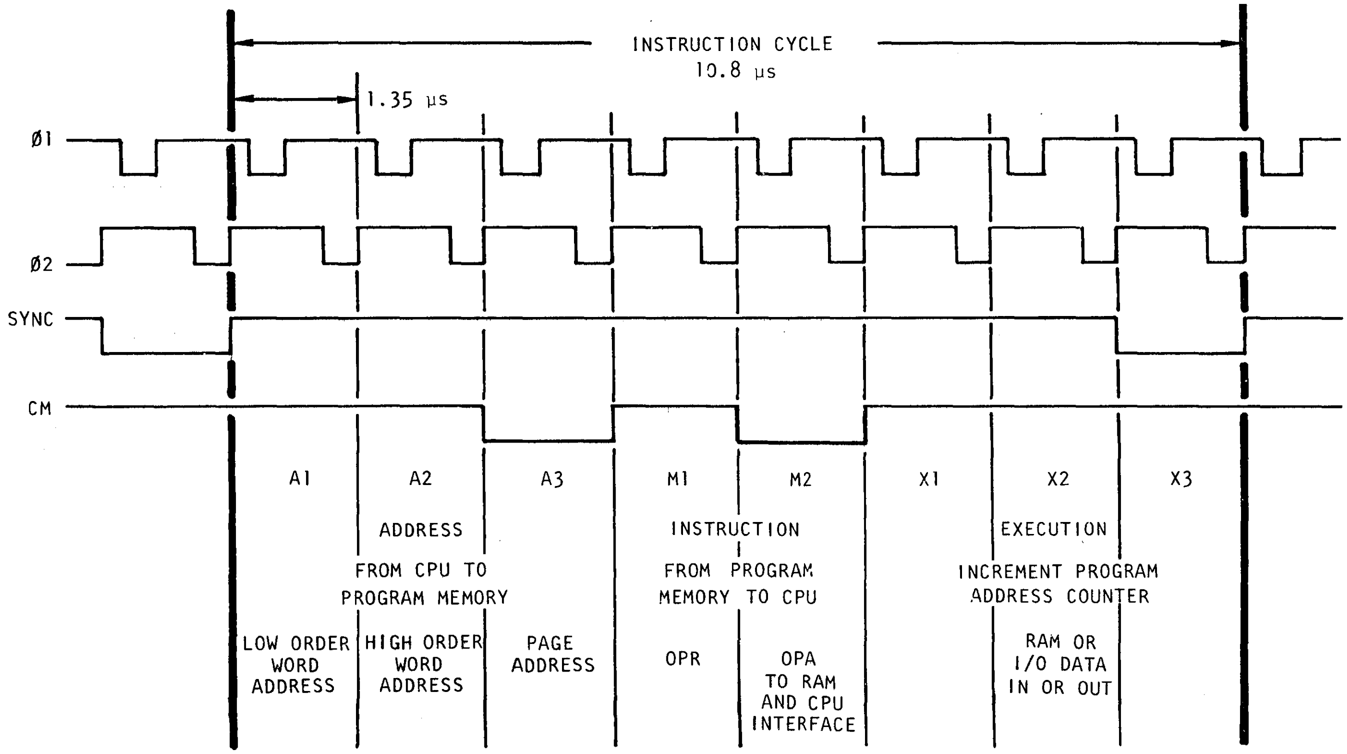


FIGURE 3-3

CPU Timing for I/O and RAM Register Instructions

DEVICE DATA SHEETS

The PLS 400 series uses 4000 series MOS devices. For exact specifications on the electrical and timing requirements of these devices refer to the Intel data sheets. As an aid to the user the 4001, 2, 3, 4 electrical specifications are shown with power supply reference of +5 and -10 volts as used in the PLS 400 system.

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias	0°C to +70°C
Input Voltages and Supply Voltage With Respect to V_{SS}	+0.5 to -20 V
Power Dissipation	1.0 W

DC AND OPERATING CHARACTERISTICS

T_A = 0°C to +70°C; V_{DD} = -10 V ±5%, V_{SS} = +5 ±5%

Logic "0" is defined as the more positive voltage (V_{IH}, V_{OH}), Logic "1" is defined as the more negative voltage (V_{IL}, V_{OL})

SUPPLY CURRENT

				Limit			
Product	Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
4001	l_{DD1}	Average Supply Current		15	20	mA	T_A = 25°C
4002	l_{DD2}	Average Supply Current		17	33	mA	T_A = 25°C
4003	l_{DD3}	Average Supply Current		5.0	8.5	mA	t_{WL} = t_{WH} = 8 μs; T_A = 25°C
4004	l_{DD4}	Average Supply Current		30	40	mA	T_A = 25°C
INPUT CHARACTERISTICS							
4001/2/3/4	l_{L1}	Input Leakage Current			10	mA	V_{IL} = V_{DD}
4001/2/3/4	V_{IH}	Input High Voltage (all inputs except clock)	+3.5		+5.3	V	
4001/2/3/4	V_{IL}	Input Low Voltage (all inputs except clock)	V_{DD} V_{DD}		-0.5 -1.5	V V	Inverting Input Noninverting Input
4001/2/4	V_{ILC}	Clock Input Low Voltage	V_{DD}		-8.4	V	
4001/2/4	V_{IHC}	Clock Input High Voltage	+3.5		+5.3	V	
4001	R_I	I/O Pins Input Resistance	10	18	35	KΩ	Internal input resistor is optional
OUTPUT CHARACTERISTICS							
4001/2/4	I_{LO}	Data Bus Output Leakage Current			10	mA	V_{OUT} = -7 V, chip disabled
4001/2/3/4	V_{OH}	Output High Voltage		5.0	4.5	V	Driving 4000 series loads only
4001/2/4	I_{OL1}	Data Lines Sinking Current "1" Level	10	18		mA	V_{OUT} = 5 V
4001/2	I_{OL2}	L/O Output Lines Sinking Current, "1" Level	2.5	5		mA	V_{OUT} = 5 V
4003	I_{OL3}	Parallel Out Pins Sinking Current, "1" Level	0.6	1.0		mA	V_{OUT} = 5 V
4003	I_{OL4}	Serial Out Sinking Current, "1" Level	1.0	2.0		mA	V_{OUT} = 5 V
4004	I_{OL5}	CM-ROM Sinking Current "1" Level	6.5	12		mA	V_{OUT} = 5 V
4004	I_{OL6}	CM-RAM Lines Sinking Current "1" Level	2.5	4		mA	V_{OUT} = 5 V
4001/2/4	V_{OL1}	Data Lines, CM Lines, Sync Output Low Voltage	-7	-5	-1.5	V	I_{OL1} = 500 μA
4003	V_{OL3}	Output Low Voltage	-6	-2.5	-1.5	V	I_{OL3} = 10 μA
4001/2	V_{OL2}	I/O Output Lines Output Low Voltage	-7	-2.5	-1.5	V	I_{OL2} = 50 μA
4001/2/4	R_{OH1}	Output Resistance Data Lines "O" Level		150	250	Ω	V_{OUT} = 4.5 V
4001/2	R_{OH2}	Output Resistance I/O Line "O" Level		1.2	1.8	KΩ	V_{OUT} = 4.5 V
4003	R_{OH2}	Parallel-Out Pins Output Resistance "O" Level		400	750	Ω	V_{OUT} = 4.5 V
4003	R_{OH4}	Serial Out Output Resistance "O" Level		650	1200	Ω	V_{OUT} = 4.5 V
4004	R_{OH5}	CM-ROM Output Resistance "O" Level		320	600	Ω	V_{OUT} = 4.5 V
4004	R_{OH6}	CM-RAM Lines Output Resistance "O" Level		1.1	1.8	KΩ	V_{OUT} = 4.5 V

Typical values are for T_A = 25°C and Nominal Supply Voltages

4. PLS 400 SYSTEM ORGANIZATION

CENTRAL PROCESSING UNIT

Central processing unit consist of a central processing unit (CPU) and a memory that has a stored sequence of instructions for the CPU. The CPU is operated by a clock circuit to alternately fetch and execute the memory instructions. The CPU fetches an instruction by sending an address from a program address counting register to the program memory. The program memory decodes the address and sends the selected instruction to the CPU. The CPU stores the instruction in an instruction register where it is decoded and executed.

PLS 400 SYSTEMS

The PLS 400 systems are controlled by the Intel 4004 CPU chip. The CPU performs control and data transfer functions with the logic elements shown in the system data flow diagram Figure 4-1. The CPU communicates with program memory, RAM registers and I/O ports by connecting appropriate elements of the system to the 4 bit CPU BUS. Conceptually the information paths exist as shown in Figure 4-1.

In addition to an instruction register and program address counter, the CPU contains a program address counter stack, an arithmetic logic unit (ALU) with a four bit accumulator register, and 16 four bit registers for intermediate data storage.

INSTRUCTION REGISTER

The instruction register shown in Figure 4-2 consists of eight bits of storage and decoding for single word and the first word of double word instructions as they are received from the program memory. The 4 bits associated with M1 time are always instruction information. The 4 bits at M2 time can be additional instruction information, data constants, or page address information.

The second word of a double word instruction does not go to the instruction register but goes as either data to the index registers or as a word address to A1 and A2 of the program address counter.

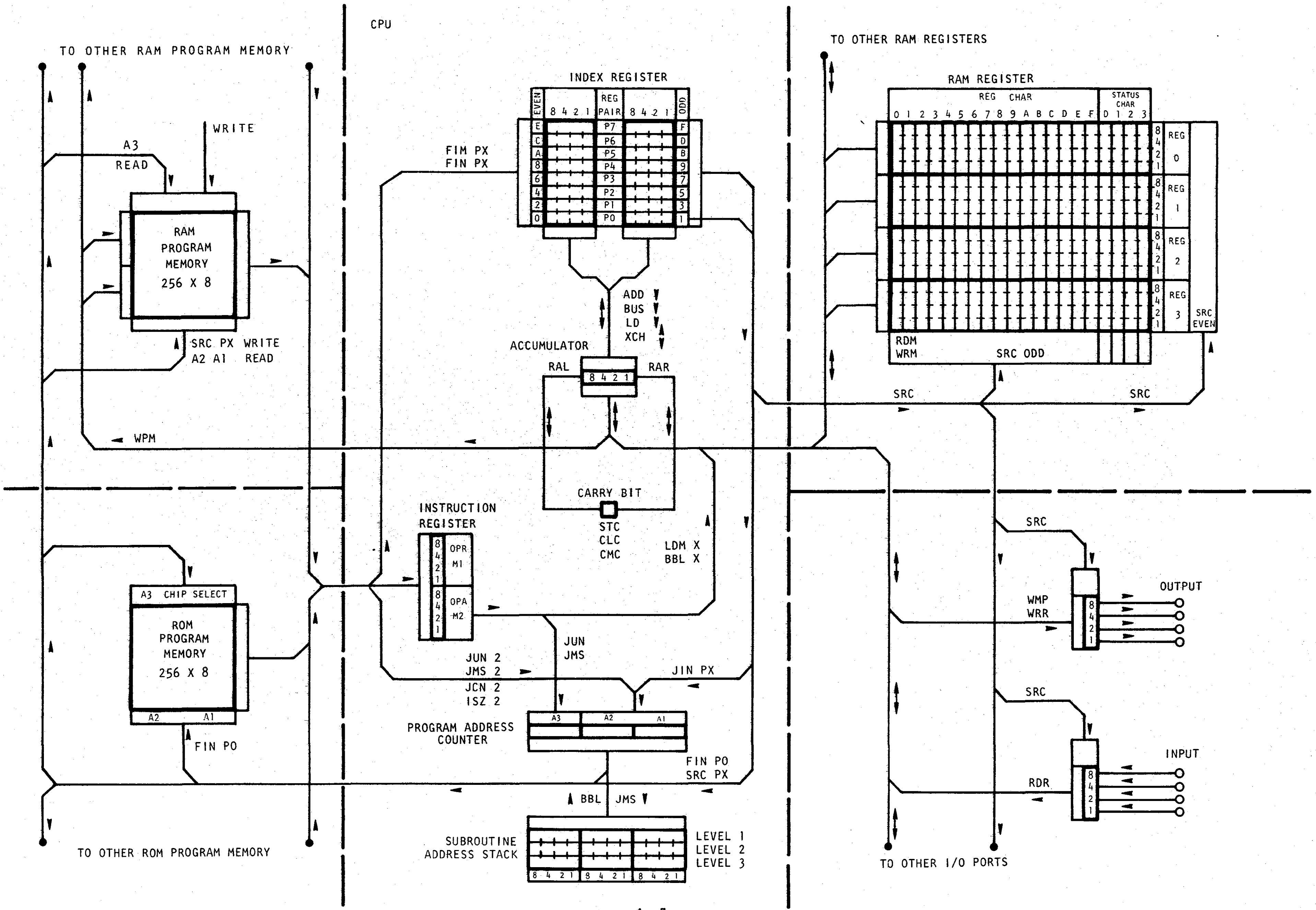


FIGURE 4-1

PLS-400 System Data Flow

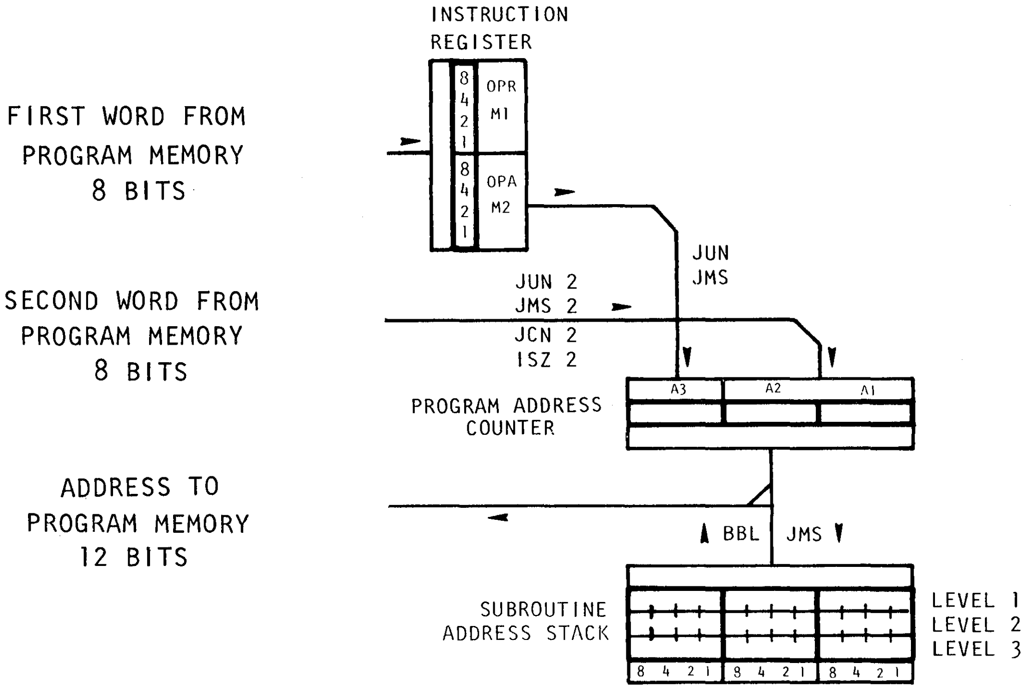


FIGURE 4-2

Instruction Register, Program Address Counter, and Subroutine Address Stack

PROGRAM ADDRESS COUNTER

The program address counter shown in Figure 4-2 is a 12 bit sequential counter which keeps track of the location of the next instruction to be executed from program memory. The four most significant bits (A3) are called the page address and the eight least significant bits (A2 and A1) are the word address of the instruction on a page. The program address counter is normally incremented by 1 for each instruction word unless the instruction is the type which modifies the count by loading a new address.

SUBROUTINE ADDRESS STACK

The subroutine address stack shown in Figure 4-2 consists of three 12 bit registers used to save the program return address for each of three allowable subroutine levels. The subroutine address stack is controlled by two CPU instructions, an entry instruction JMS and a return instruction BBL. Each entry to a subroutine causes the program address counter to be transferred to the top most level of the subroutine address stack. The three levels in turn are pushed-down to accommodate the new entry. The lowest level is lost off the bottom of the stack. Each return from a subroutine causes the stack to be pulled-up one level with the top most address going to the program address counter.

INDEX REGISTERS

The index registers consist of sixteen 4 bit registers which can be directly operated on by various instructions, either individually or in pairs. Figure 4-3 shows the registers organized as the even numbered and the odd numbered registers, or as seven pairs, each pair consisting of one even and one odd numbered register.

When the registers are being used with the 4 bit accumulator by various instructions they are used individually. When data is loaded direct from program memory or the registers are used for address control they are used in pairs because of the 8 bit requirement for these functions.

ARITHMETIC LOGIC UNIT

The arithmetic logic unit consists of a 4 bit accumulator register and a carry flip-flop as shown in Figure 4-3. In addition to providing the arithmetic functions of ADD and SUBtract the accumulator is the central control and distribution point for data flow in the system. All data transfers to and from I/O. RAM registers and the index register occurs with the accumulator register.

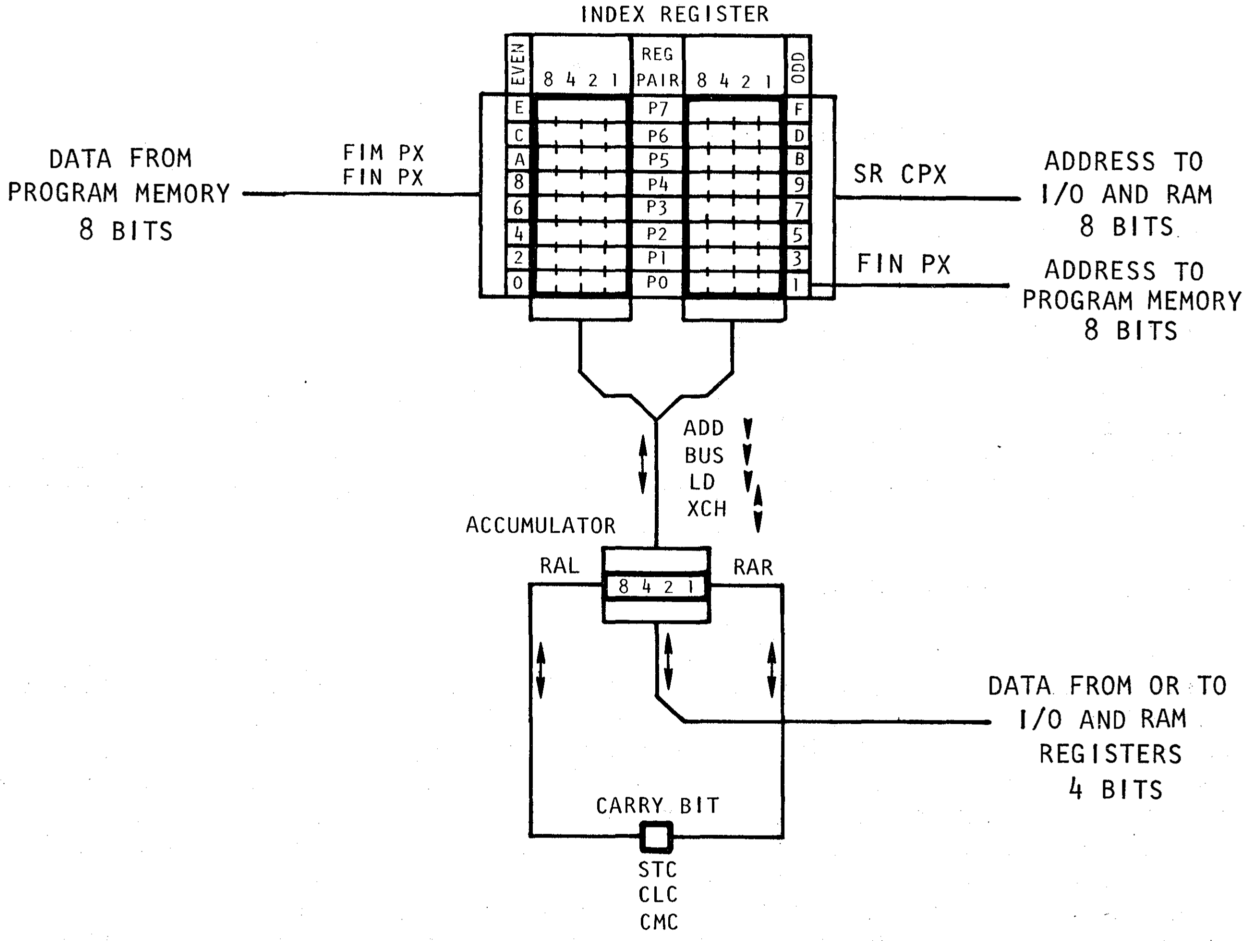


FIGURE 4-3
Index Registers

In addition to the instructions which control data transfer to or from the accumulator there are instructions which directly control the accumulator or its associated carry bit. The accumulator can be tested, incremented, decremented, set to any value, cleared, complemented, rotated right or left through the carry besides being manipulated for decimal arithmetic. The carry bit can be set, cleared, complemented, or tested.

PROGRAM MEMORY

Program memory stores the instruction to be executed by the CPU and is defined by the CPU instruction set as a page oriented memory of 256 words per page as shown in Figure 4-4. The CPU addresses the page and word and the program memory sends the 8 bit word at that address to the CPU.

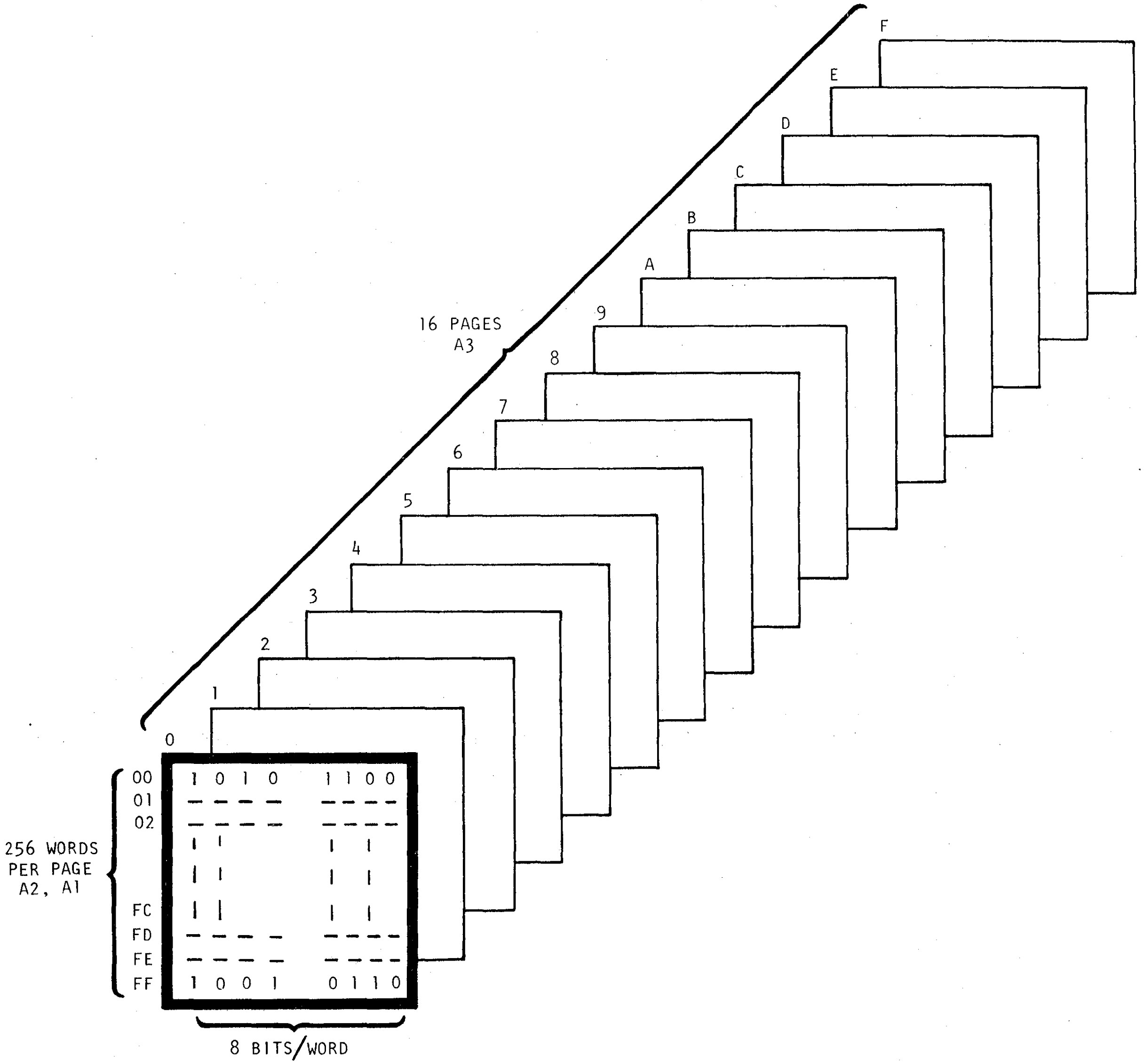


FIGURE 4-4

Organization of Program Memory as Defined by the CPU Instructions

The 12 bit addressing capability of the CPU allows direct access to 16 pages with the four A3 bits used as page address. The eight bits at A1 and A2 are used for the word address within a page. It is important to understand the page organization in terms of the address control instructions (jumping and branching). Certain address instructions use the full twelve bits of address and may be used to change control within a page or from page to page. Other address control instructions use only eight bits of address and are limited to changing control only within a page.

The PLS 400 systems are implemented with ROM (read only memory) program memory only. In addition the PLS 403 has all the control lines available for implementing RWM (read write memory) program memory. ROM program memory is used for systems in fixed applications. RWM memory is used where it is desired to change the system application by the operator. RWM is a considerable step in system complexity in hardware and programs, and is therefore not recommended unless absolutely necessary.

ROM PROGRAM MEMORY

ROM program memory on the PLS 400 system is accomplished as shown in Figure 4-4 using programmable erasable ROMs organized as 256 location of 8 bits. This organization is equivalent to the page size of the CPU therefore each ROM chip equals one page. Other ROM sizes and organizations can be used if the appropriate hardware addressing is provided.

ROM program memory addressing is an automatic function of the PLS hardware. The only control the program designer has over ROM memory is use of the program control instructions to change the instruction sequence.

RWM PROGRAM MEMORY

RWM program memory can be accomplished on the PLS 403 system only. The Intel 4008 and 4009 interface devices provide the address lines and control lines necessary for writing into the desired memory type. The WPM instruction allows writing 4 bits at a time from the accumulator to the RWM. For applications and suggested implementation, refer to the Intel 4008 and 4009 data sheet.

RAM REGISTER STORAGE

The PLS 400 systems use the Intel 4002 RAM register devices for program controlled data storage. Each 4002 is organized as four registers of 20 characters as shown in Figure 4-5. Each 20 character register consists of 16 individually addressable characters of main storage plus 4 instruction selectable status characters.

The instruction capability of the CPU allows addressing of up to 32 of the 4002 RAM devices. This is accomplished through an organization of 8 banks of 4 RAM chips per bank. RAM banks are selected by the DCL instruction that specifies which of the four CM-RAM lines out of the CPU will be active. The active CM-RAM line designates which RAM bank will respond to the SRC instruction. The SRC instruction selects the RAM chip, register and character. A summary of RAM addressing is given in Table 4-1 and further definition of RAM addressing is given in Section 6 under the SRC and DCL instructions.

TABLE 4-1

RAM Addressing

Level	Instruction
RAM Bank	DCL
RAM Chip	SRC, Even Register high order bits
RAM Register	SRC, Even Register low order bits
RAM Character	SRC, Odd Register 4 bits

INPUTS AND OUTPUTS

The flow of data into and out of the PLS systems is accomplished through the I/O ports of four lines each. To accomplish an input or output function a port must first be addressed by the CPU instruction SRC. The even register of the SRC pair contains the address of the port to be selected. Once a port has been addressed it remains selected for as many input or output operations as desired until another port is addressed.

There are two types of output ports and one type of input port. Each RAM register device has an output port packaged physically within the device. This port shares chip select addressing with the RAM but has its own instruction WMP for the transfer of data from the accumulator to the port. The port latches any data sent to it and retains it as a stable output until a subsequent WMP instruction changes the data. The RAM port lines are MOS low level active outputs capable of driving one low power TTL load.

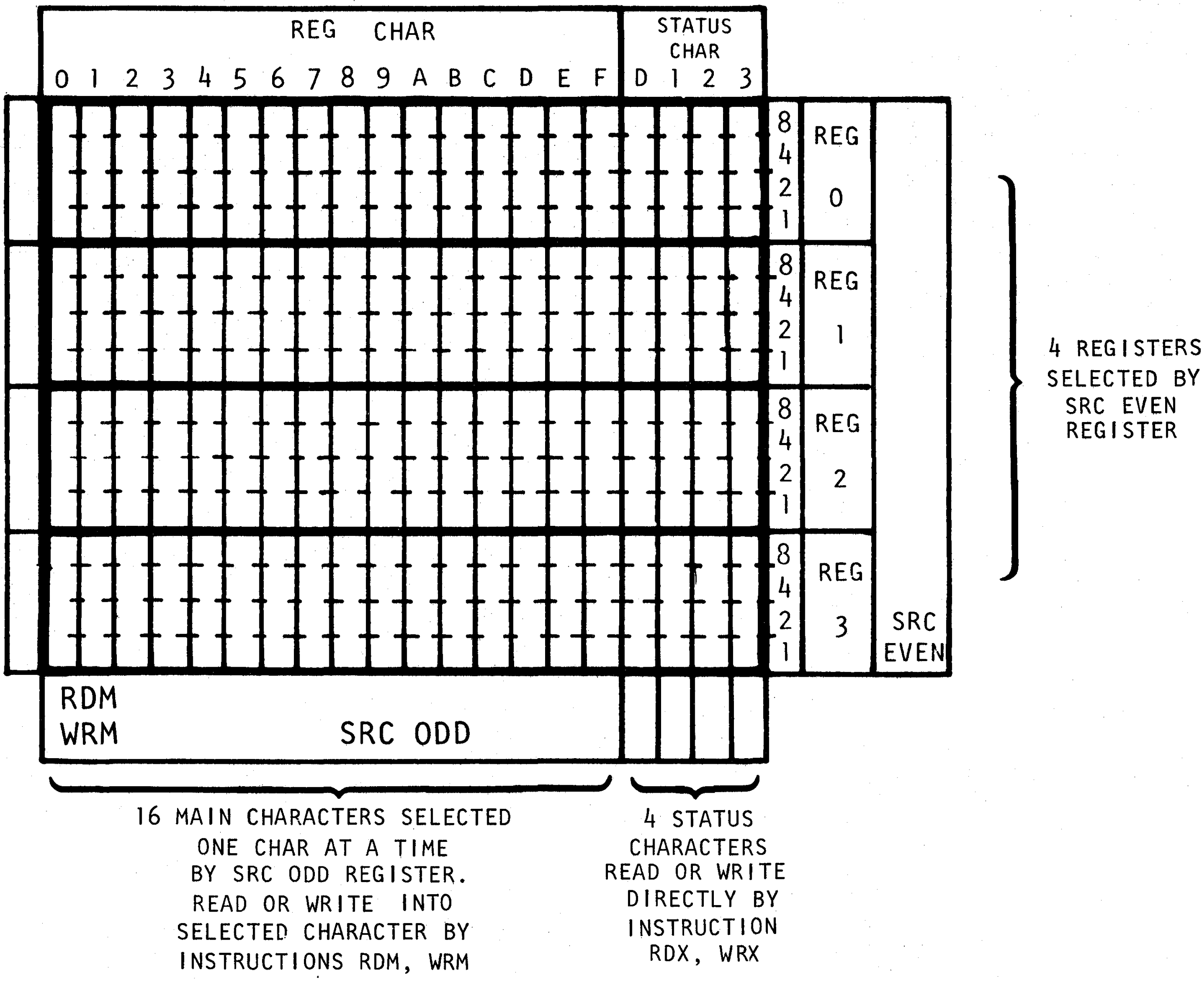

REG CHAR
0 1 2 3 4 5 6 7 8 9 A B C D E F
STATUS CHAR
D 1 2 3
8 4 2 1
REG 0
REG 1
REG 2
REG 3
SRC EVEN
4 REGISTERS SELECTED BY SRC EVEN REGISTER
RDM
WRM
SRC ODD
16 MAIN CHARACTERS SELECTED ONE CHAR AT A TIME BY SRC ODD REGISTER. READ OR WRITE INTO SELECTED CHARACTER BY INSTRUCTIONS RDM, WRM
4 STATUS CHARACTERS READ OR WRITE DIRECTLY BY INSTRUCTION RDX, WRX

FIGURE 4-5

RAM Index Register

The other type of output port is implemented in the PLS 400 systems using TTL logic. The CPU instruction WRR is used to send data to a TTL quad D type flip-flop from the accumulator. The TTL flip-flops latch the data as a stable output until a subsequent WRR instruction changes the data.

The PLS 400 input ports are also implemented with TTL logic. The CPU instruction RDR reads data from the selected input port into the accumulator.

HEXADECIMAL NOTATION

The basic 4 bit structure of the CPU makes it convenient to use hexadecimal notation to express with a single character, one-of-sixteen possible combinations.

The single hexadecimal character notation 0 → 9, A → F is used to refer to the:

- 16 Basic Instructions
- 16 I/O and RAM Instructions
- 16 Accumulator Instructions
- 16 Index Registers
- 16 Pages of Program Memory Capacity
- 16 RAM Register Chip Capacity
- 16 Characters in a RAM Register
- 16 Output Ports
- 16 Input Ports

A double hexadecimal character notation is applied to the 8 bit instruction word address for program memory, where the decimal addresses 000 through 255 are given as 00 through FF in hexadecimal.

Table 4-2 shows the hexadecimal notation for sixteen combinations. Additional hexadecimal tables are given in the appendix.

TABLE 4-2

Hexadecimal Notation for Sixteen Combinations

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

5. INSTRUCTION TABLE

This section presents the 4004 CPU instructions in a short table form. Section 6 contains detailed descriptions of the instructions.

HEX CODING	MNEMONIC		DESCRIPTION OF OPERATION
	OPR	OPA	
0 0	NOP		No operation.
1 C_x A_2 A_1	JCN	C_x LABEL	Jump on condition C_x to the program memory address A_1, A_2, otherwise continue in sequence.
2 P_xe D_2 D_1	FIM D_2	P_xO D_1	Fetch immediate from program memory data D_1, D_2 to index register pair P_x
2 P_xo	SRC	P_x1	Send register control. Send the contents of index register pair P_x to I/O ports and RAM register as chip select and RAM character address.
3 P_xe	FIN	P_xO	Fetch indirect. Send contents of register pair 0 out as a program memory address. Data fetched is placed into register pair P_x
3 P_xo	JIN	P_x1	Jump indirect. Jump to the program memory address designated by contents of register pair P_x
4 A_3 A_2 A_1	JUN	LABEL	Jump unconditional to program memory address A_1, A_2, A_3.
5 A_3 A_2 A_1	JMS	LABEL	Jump to subroutine located at program memory address A_1, A_2, A_3. Save previous address (push down in stack).
6 R_x	INC	R_x	Increment contents of register R_x.
7 R_x A_2 A_1	ISZ	R_x LABEL	Increment and skip on zero. Increment contents of register R_x, if result is not 0 go to program memory address A_1, A_2, otherwise skip to the next instruction in sequence.
8 R_x	ADD	R_x	Add contents of register R_x to accumulator.
9 R_x	SUB	R_x	Subtract contents of register R_x to accumulator with borrow.
A R_x	LD	R_x	Load contents of register R_x to accumulator.
B R_x	XCH	R_x	Exchange contents of index register R_x and accumulator.
C D_1	BBL	D_1	Branch back one level in stack to the program memory address stored by a prior JMS instruction. Load data D_1 to accumulator.
D D_1	LDM	D_1	Load data D_1 to accumulator.
E X	I/O and RAM register instructions		
F X	Accumulator instructions		

A_1	Low order address bits
A_2	High order address bits
A_3	Chip select
P_xO	Register P_O through P_7 designated by odd characters 1, 3, 5, 7, 9, B, D, F
P_x1	Register P_O through P_7 designated by even characters 0, 2, 4, 6, 8, A, C, E

R_x	Register 0 → F
D_1	Data character #1
D_2	Data character #2
C_x	Jump conditions

I/O AND RAM REGISTER INSTRUCTIONS

HEX CODING	MNEMONIC		DESCRIPTION OF OPERATION
	OPR	OPA	
E 0	WRM		Write the contents of the accumulator into the previously selected RAM register character.
E 1	WMP		Write the contents of the accumulator into the previously selected RAM output port. (Output lines.)
E 2	WRR		Write the contents of the accumulator into the previously selected output port. (I/O lines.)
E 3	WPM		Write the contents of the accumulator into the previously selected RAM program memory.
E 4	WRO		Write the contents of the accumulator into the previously selected RAM status character 0.
E 5	WR1		Write the contents of the accumulator into the previously selected RAM status character 1.
E 6	WR2		Write the contents of the accumulator into the previously selected RAM status character 2.
E 7	WR3		Write the contents of the accumulator into the previously selected RAM status character 3.
E 8	SBM		Subtract the previously selected RAM register character from accumulator with borrow.
E 9	RDM		Read the previously selected RAM register character into the accumulator.
E A	RDR		Read the contents of the previously selected input port into the accumulator. (I/O lines.)
E B	ADM		Add the previously selected RAM register character to accumulator with carry.
E C	RDO		Read the previously selected RAM status character 9 into accumulator.
E D	RD1		Read the previously selected RAM status character 1 into accumulator.
E E	RD2		Read the previously selected RAM status character 2 into accumulator.
E E	RD3		Read the previously selected RAM status character 3 into accumulator.

ACCUMULATOR INSTRUCTIONS

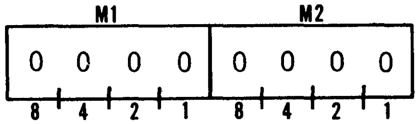
HEX CODING	MNEMONIC		DESCRIPTION OF OPERATION
	OPR	OPA	
F 0	CLB		Clear both. (Accumulator and carry.)
F 1	CLC		Clear carry.
F 2	IAC		Increment accumulator.
F 3	CMC		Complement carry.
F 4	CMA		Complement accumulator.
F 5	RAL		Rotate left. (Accumulator and carry.)
F 6	RAR		Rotate right. (Accumulator and carry.)
F 7	TCC		Transmit carry to accumulator and clear carry.
F 8	DAC		Decrement accumulator.
F 9	TCS		Transfer carry subtract and clear carry.
F A	STC		Set carry.
F B	DAA		Decimal adjust accumulator.
F C	KBP		Keyboard process. Converts the contents of the accumulator from a one out of four code to a binary code.
F D	DCL		Designate command line.
F E			
F F			

Condition Table for ICN Instruction

JCN HEX	C_x MNEMONIC	C_8	C_4	C_2	C_1	Invert Jump Condition Jump if Accumulator = 0 Jump if Carry Bit = 0 Jump if Test Input = 0
10		0	0	0	0	NO OPERATION
11	TO	0	0	0	1	Jump if test = 0
12	C1	0	0	1	0	Jump if CY = 1
13	TO+C1	0	0	1	1	Jump if test = 0 or CY = 1
14	AO	0	1	0	0	Jump if AC = 0
15	TO+AO	0	1	0	1	Jump if test = 0 or AC = 0
16	C1+AO	0	1	1	0	Jump if CY = 1 or AC = 0
17	TO+C1+AO	0	1	1	1	Jump if test = 0 or CY = 1 or AC = 0
18		1	0	0	0	Jump Unconditionally
19	T1	1	0	0	1	Jump if test = 1
1A	CO	1	0	1	0	Jump if CY = 0
1B	T1CO	1	0	1	1	Jump if test = 1 and CY = 0
1C	A1	1	1	0	0	Jump if AC ≠ 0
1D	T1A1	1	1	0	1	Jump if test = 1 and AC ≠ 0
1E	COA1	1	1	1	0	Jump if CY = 0 and AC ≠ 0
1F	T1COA1	1	1	1	1	Jump if test = 1 and CY = 0 and AC ≠ 0

6. INSTRUCTION DESCRIPTIONS

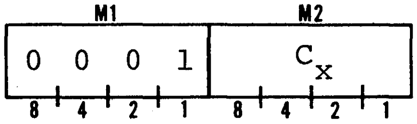
NO OPERATION NOP



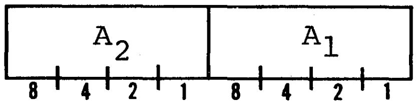
0 0

No operation is performed by this instruction except that the program address counter counts to the next instruction address in sequence. This instruction can be used as a one cycle time delay. To avoid problems with power-on reset, the first instruction at program address 000 should always be an NOP.

JUMP ON CONDITION JCN



First Word **1** C_x



Second Word A_2 A_1

If the designated condition (C_x) is true, program control is transferred to the instruction located at the 8 bit address A_2, A_1 of the current page, otherwise program control continues in sequence. If the JCN occupies the last two positions of a page or overlaps the page boundary, program control is transferred to the 8 bit address on the next page in sequence.

JCN is one of the two decision making instructions of the CPU, the other being ISZ. JCN allows a decision on the following tests:

Test accumulator for zero or nonzero

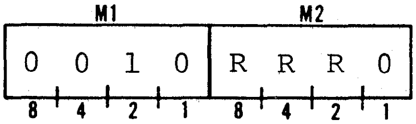
Test carry bit for logic one or zero

Test external input lead for high or low

Table 5 provides detailed definitions of conditions C_x.

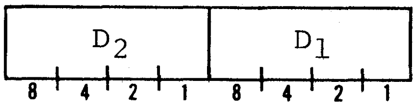
FETCH IMMEDIATE

FIM



First Word

$2 \; P_x0$



Second Word

$D_2 \; D_1$

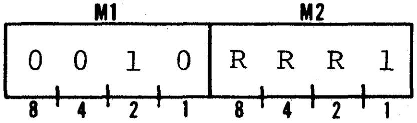
Load the 8 bits of data from the second word D_2, D_1 to the designated pair of index registers P_x0.

FIM uses the even register numbers to designate a pair. The only valid operand codes for P_x0 are 0, 2, 4, 6, 8, A, C, and E. FIM provides the most efficient way to initialize a pair of index registers.

RRR defines one of the eight register pairs P0 through P7. The 0 following RRR is part of the command decoding and distinguishes the FIM from the SRC.

SEND REGISTER CONTROL

SRC



$2 \; P_x1$

Send the contents of index register pair P_x1 to the I/O ports and RAM registers as chip select and/or RAM character select. SRC uses the odd register numbers to designate a pair. The only valid operand codes for P_x1 are 1, 3, 5, 7, 9, B, D, and F.

RRR defines one of the eight register pairs P0 through P7. The 1 following RRR is part of the command decoding and distinguishes the SRC from the FIM.

It is necessary to address the I/O port or RAM register character using an SRC instruction before an I/O operation or a RAM register operation can be performed. The same SRC instruction can be used to address both I/O ports and RAM registers, however, the meaning of the address in the designated pair P_x1 is different for each as shown below.

The I/O port is addressed by the contents of the even register designated by P_x. The odd register does not serve any purpose in selecting I/O ports.

ELEMENT ADDRESSED	REGISTER PAIR P_x	
	EVEN REGISTER	ODD REGISTER
I/O PORT	PORT SELECT 8 4 2 1	NOT USED 8 4 2 1
RAM REGISTER	CHIP SELECT | REGISTER SELECT 8 4 2 1	CHARACTER 8 4 2 1

The RAM chip select is addressed by the high order 2 bits of the even register, the RAM register within the selected chip is addressed by the low order 2 bits of the even register and the character within the RAM register is addressed by the 4 bits of the odd register.

Addressing of the I/O port and RAM registers by the even register is tabulated in Table 6-1. The table covers any one bank of RAM registers. To select other RAM banks refer to the DCL instructions.

TABLE 6-1

I/O Port and RAM Selection for One Bank by Even Register Contents, as Used with SRC Instruction

CONTENTS OF EVEN REGISTER	I/O PORT SELECTED	RAM # AND RAM REGISTER SELECTED			
		RAM #	REGISTER	RAM DEVICE TYPE	RAM PIN 10 WIRED
0	0	0	0	4002-1	HIGH
1	1	0	1		
2	2	0	2		
3	3	0	3		
4	4	1	0	4002-1	LOW
5	5	1	1		
6	6	1	2		
7	7	1	3		
8	8	2	0	4002-2	HIGH
9	9	2	1		
A	A	2	2		
B	B	2	3		
C	C	3	0	4002-2	LOW
D	D	3	1		
E	E	3	2		
F	F	3	3		

FETCH INDIRECT

FIN

M1				M2			
0	0	1	1	R	R	R	0
8	4	2	1	8	4	2	1

3 P_X0

The 8 bit content of register pair 0 is sent out as an address to the current page of program memory. The 8 bit word at that location is loaded as data into the designated index register pair P_X0. If the FIN occupies the last position of a page, data will be fetched from the next page in sequence. The program counter is not affected.

After the FIN has been executed the next instruction in sequence will be addressed. However, the FIN is a one word instruction, and it requires an additional instruction cycle to retrieve the 8 bits of data for the designated register pair. This extra cycle must be considered when the FIN is used in routines with timing considerations.

FIN uses the even register numbers to designate a pair. The only valid operand codes for P_X0 are 0, 2, 4, 6, 8, A, C, and E. The FIN instruction is useful for retrieving data from look-up or translation tables.

RRR defines one of the eight register pairs P0 through P7. The 0 following RRR is part of the command decoding and distinguishes the FIN from the JIN.

JUMP INDIRECT

JIN

M1				M2			
0	0	1	1	R	R	R	1
8	4	2	1	8	4	2	1

3 P_X1

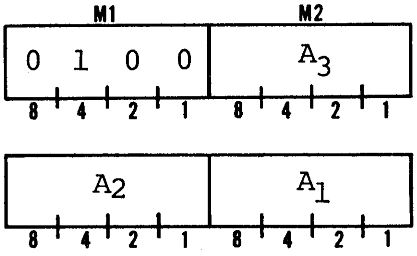
The 8 bit content of the designated register pair P_X1 is loaded into the low order 8 positions of the program address counter. Program control is transferred to the instruction at that address on the same page. If the JIN occupies the last position of the current page program control transfers to the 8 bit address of the next page in sequence.

The 8 bit content of the register pair is not affected.

JIN uses the odd register numbers to designate the pair P_X. The only valid operand codes for P_X1 are 1, 3, 5, 7, 9, B, D, and F.

RRR defines one of the eight register pairs P0 through P7. The 1 following RRR is part of the command decoding and distinguishes the JIN from the FIN.

JUMP UNCONDITIONAL JUN

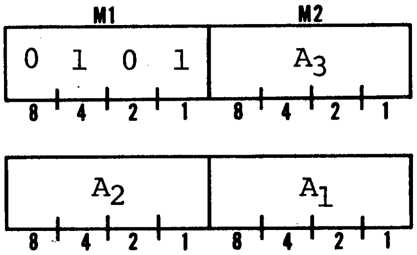


First Word $4\ A_3$

Second Word $A_2\ A_1$

Program control is unconditionally transferred to the instruction located at the address A_3, A_2, and A_1. The CPU accomplishes this internally by transferring A_3 from the operand of the instruction register and A_2, A_1 from program memory to the program address counter.

JUMP TO SUBROUTINE JMS



First Word $5\ A_3$

Second Word $A_2\ A_1$

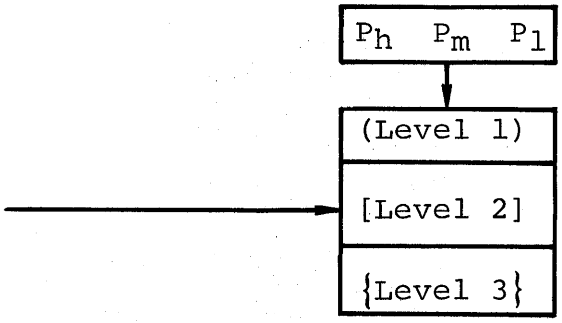
The subroutine address stack is pushed down one level. The program address counter, containing the 12 bit address of the instruction following the second word of the JMS, is transferred to the topmost stack level. Program control is transferred to the instruction located at A_3, A_2, and A_1 from program memory to the program address counter.

First word. In Instruction Register 5 A_3

Second word. From Program Memory A_2 A_1

Program Address Counter

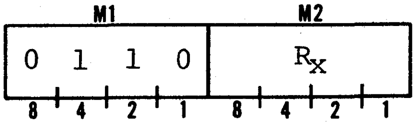
Subroutine Address Stack



Stack shown fully loaded

INCREMENT REGISTER

INC



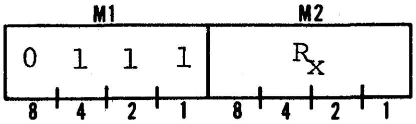
6 R_x

The 4 bit content of the designated register R_x is incremented by 1. If the count causes the register to overflow, the register is set to zero.

The carry bit and accumulator are not affected.

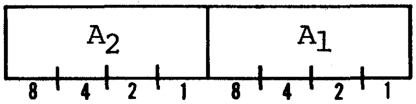
INCREMENT REGISTER SKIP IF ZERO

ISZ



First Word

7 R_x



Second Word

A_2 A_1

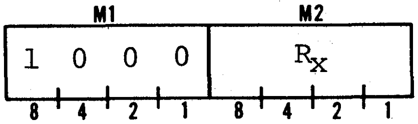
The contents of the designated register R_x is incremented by 1. If the result is zero, program control continues in sequence. If the result is not zero, program control is transferred to the instruction located at the 8 bit address A_2, A_1 on the same page. If the ISZ occupies the last two positions of a page, or overlaps the page boundary, program control is transferred to the 8 bit address on the next page in sequence.

The accumulator and carry are not affected.

ISZ is one of the two decision making in structions of the CPU. The other is JCN. ISZ allows a program control decision to be made based on the count of a register. Examples of ISZ use may be found in Section 8 of this manual.

ADD REGISTER TO ACCUMULATOR

ADD



8 R_x

The 4 bit content of the designated index register R_x is added to the contents of the accumulator with carry. The result is stored in the accumulator. The carry is set to 1 if a sum greater than 15 was generated, otherwise the carry is set to 0.

The contents of the index register is not affected.

Accumulator in		a_8 a_4 a_2 a_1	Augend
Carry in		C_{in}	Carry in
Register	+)	R_8 R_4 R_2 R_1	Addend
Accumulator out		a_8 a_4 a_2 a_1	Sum
Carry out		C_{out}	Carry out

Addition of words longer than 4 bits (multiple precision addition) may be accomplished by starting with the LSD, working on 4 bits at a time until the desired word length has been operated on. It is important not to modify the carry bit between each 4 bits.

SUBTRACT REGISTER FROM ACCUMULATOR SUB

M1				M2			
1	0	0	1	R_x			
8	4	2	1	8	4	2	1

9 R_x

The 4 bit content of the designated register R_x is subtracted from the accumulator with borrow. The result is stored in the accumulator. If a borrow is generated, (i.e., R_x > accumulator) the carry bit is set to 0; is a borrow is not generated the carry is set to 1.

The content of the index register R_x is not affected.

The CPU performs the subtraction by adding the complement of the index register plus the complement of the carry to the accumulator.

Accumulator in		a_8 a_4 a_2 a_1	Minuend
Carry in		$\overline{C_{in}}$	Borrow in
Register	+)	$\overline{R_8}$ $\overline{R_4}$ $\overline{R_2}$ $\overline{R_1}$	Subtrahend
Accumulator out		a_8 a_4 a_2 a_1	Result
Carry out		C_{out}	Borrow out

Subtraction of words longer than 4 bits (multiple precision subtraction) can be accomplished by starting with the LSD, working on 4 bits at a time until the desired word length has been operated on. It is required that the carry bit be complemented between each 4 bits for the correct result.

The SUB instruction is useful for performing a compare function. The compare is performed by initially clearing the carry bit and subtracting the 4 bit word R_x to be compared from the accumulator.

The conditions to be tested for comparison results following subtraction are presented below:

COMPARISON	ACCUMULATOR	CARRY	MNEMONIC TEST CONDITION
REG > ACC	≠ 0	0	CO
REG = ACC	0	1	AO
REG < ACC	≠ 0	1	A1·C1
REG ≦ ACC	X	1	C1
REG ≠ ACC	≠ 0	X	A1

LOAD REGISTER TO ACCUMULATOR — LD

M1	M2
1 0 1 0	R_x
8 4 2 1	8 4 2 1

A R_x

The 4 bit content of the designated index register R_x is loaded into the accumulator. The previous contents of the accumulator are lost.

The content of the index register and the carry bit are not affected.

EXCHANGE REGISTER WITH ACCUMULATOR — XCH

M1	M2
1 0 1 1	R_x
8 4 2 1	8 4 2 1

B R_x

The 4 bit content of the designated index register R_x is loaded into the accumulator. The prior content of the accumulator is loaded into the designated register R_x.

The carry bit is not affected.

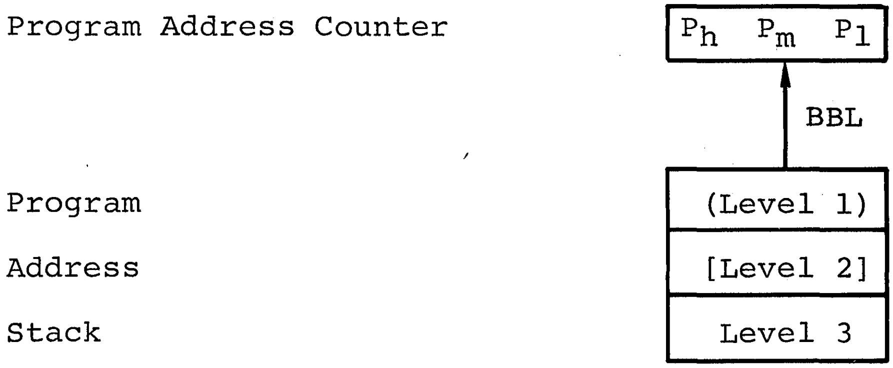
This is the only instruction which allows the accumulator to be loaded into an index register.

BRANCH BACK AND LOAD ACCUMULATOR — BBL

M1	M2
1 1 0 0	D_x
8 4 2 1	8 4 2 1

C D_x

BBL is used to return from subroutine to main program. The subroutine address stack is pulled up one level. The top-most address is placed in the program address counter causing program control to be transferred to the sequential instruction following the previous JMS.



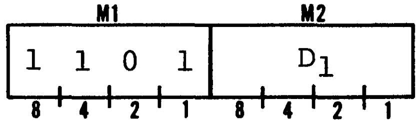
Stack shown fully loaded

The 4 bits of data D_x in the operand portion of the instruction are loaded into the accumulator. The previous accumulator data is lost.

The carry bit is not affected.

LOAD DATA TO ACCUMULATOR

LDM



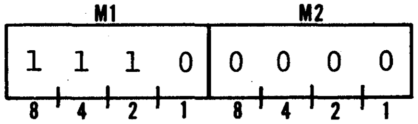
D D_1

The 4 bits of data D_1 stored in the operand field of the instruction word are loaded into the accumulator. The previous contents of the accumulator are lost.

The carry bit is not affected.

WRITE ACCUMULATOR INTO RAM CHARACTER

WRM



E0

The accumulator content is written into the RAM main memory character location previously selected by an SRC instruction.

The accumulator and carry are not affected.

WRITE MEMORY PORT

WMP

M1				M2			
1	1	1	0	0	0	0	1
8	4	2	1	8	4	2	1

E1

The content of the accumulator is transferred to the RAM output port previously selected by an SRC instruction. The data is available on the output pins until a new WMP is executed on the same RAM chip.

The accumulator and carry are not affected.

WRITE ROM PORT

WRR

M1				M2			
1	1	1	0	0	0	1	0
8	4	2	1	8	4	2	1

E2

The content of the accumulator is transferred to the output port previously selected by an SRC instruction. The data is available on the output pins until a new WRR is executed on the same port.

The accumulator and carry are not affected.

WRITE TO PROGRAM MEMORY

WPM

M1				M2			
1	1	1	0	0	0	1	1
8	4	2	1	8	4	2	1

E3

This instruction is used to write data into RAM program memory 4 bits at a time. The WPM instruction must be executed twice for each 8 bit RAM program memory location.

Program memory page select lines are forced to 1111. The previous SRC address is sent out on the program memory address bus and the accumulator contents becomes available as 4 bits of data on the I/O bus. Two control lines from the CPU interface circuitry control writing into the RAM.

The WPM instruction is not applicable to PLS 401 and PLS 402 systems since the program memory address bus is not available to the user. The PLS 403 configuration provides all necessary lines for implementing RAM program memory.

WRITE INTO RAM STATUS CHARACTER 0

WRO

E4

M1				M2			
1	1	1	0	0	1	0	0
8	4	2	1	8	4	2	1

The content of the accumulator is written into the RAM status character 0 previously selected by an SRC instruction.

The accumulator and carry are not affected.

WRITE INTO RAM STATUS CHARACTER 1

WR1

E5

M1				M2			
1	1	1	0	0	1	0	1
8	4	2	1	8	4	2	1

The content of the accumulator is written into the RAM status character 1 previously selected by an SRC instruction.

The accumulator and carry are not affected.

WRITE INTO RAM STATUS CHARACTER 2

WR2

E6

M1				M2			
1	1	1	0	0	1	1	0
8	4	2	1	8	4	2	1

The content of the accumulator is written into the RAM status character 2 previously selected by an SRC instruction.

The accumulator and carry are not affected.

WRITE INTO RAM STATUS CHARACTER 3

WR3

E7

M1				M2			
1	1	1	0	0	1	1	1
8	4	2	1	8	4	2	1

The content of the accumulator is written into the RAM status character 3 previously selected by an SRC instruction.

The accumulator and carry are not affected.

SUBTRACT FROM MEMORY WITH BORROW

SBM

M1				M2			
1	1	1	0	1	0	0	0
8	4	2	1	8	4	2	1

E8

The content of the RAM character previously selected by an SRC instruction is subtracted from the accumulator with borrow.

The RAM character is unaffected.

READ RAM CHARACTER

ROM

M1				M2			
1	1	1	0	1	0	0	1
8	4	2	1	8	4	2	1

E9

The content of the RAM character is transferred to the accumulator.

The carry is not affected. The 4 bit data in memory is unaffected.

READ ROM PORT

RDR

M1				M2			
1	1	1	0	1	0	1	0
8	4	2	1	8	4	2	1

EA

The data present at the input lines of the port previously selected by an SRC instruction is transferred to the accumulator.

The carry is not affected.

ADD FROM MEMORY WITH CARRY

ADM

M1				M2			
1	1	1	0	1	0	1	1
8	4	2	1	8	4	2	1

EB

The content of the RAM character previously selected by an SRC instruction is added to the accumulator with carry.

The RAM character is not affected.

READ RAM STATUS CHARACTER 0

RDO

M1				M2			
1	1	1	0	1	1	0	0
8	4	2	1	8	4	2	1

EC

The 4 bits of status character 0 of the RAM register previously selected by an SRC instruction are transferred to the accumulator.

The carry and the status character are not affected.

READ RAM STATUS CHARACTER 1

RD1

ED

M1				M2			
1	1	1	0	1	1	0	1
8	4	2	1	8	4	2	1

The 4 bits of status character 1 of the RAM register previously selected by an SRC instruction are transferred to the accumulator.

The carry and the status character are not affected.

READ RAM STATUS CHARACTER 2

RD2

EE

M1				M2			
1	1	1	0	1	1	1	0
8	4	2	1	8	4	2	1

The 4 bits of status character 2 of the RAM register previously selected by an SRC instruction are transferred to the accumulator.

The carry and the status character are not affected.

READ RAM STATUS CHARACTER 3

RD3

EF

M1				M2			
1	1	1	0	1	1	1	1
8	4	2	1	8	4	2	1

The 4 bits of status character 3 of the RAM register previously selected by an SRC instruction are transferred to the accumulator.

The carry and the status character are not affected.

CLEAR BOTH

CLB

F0

M1				M2			
1	1	1	1	0	0	0	0
8	4	2	1	8	4	2	1

Set accumulator and carry to 0.

CLEAR CARRY

CLC

F1

M1				M2			
1	1	1	1	0	0	0	1
8	4	2	1	8	4	2	1

Set carry to 0.

The accumulator is not affected.

INCREMENT ACCUMULATOR

IAC

M1				M2			
1	1	1	1	0	0	1	0
8	4	2	1	8	4	2	1

F2

The content of the accumulator is incremented by 1. No overflow sets the carry to 0; overflow sets the carry to a 1.

COMPLEMENT CARRY

CMC

M1				M2			
1	1	1	1	0	0	1	1
8	4	2	1	8	4	2	1

F3

The carry content is complemented.

The accumulator is not affected.

COMPLEMENT ACCUMULATOR

CMA

M1				M2			
1	1	1	1	0	1	0	0
8	4	2	1	8	4	2	1

F4

The content of the accumulator is complemented.

The carry is not affected.

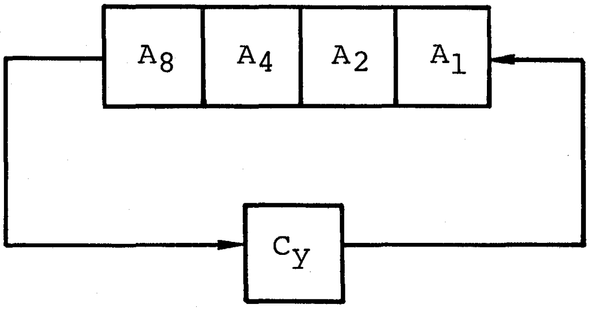
ROTATE LEFT

RAL

M1				M2			
1	1	1	1	0	1	0	1
8	4	2	1	8	4	2	1

F5

The content of the accumulator and carry are rotated left one bit position.



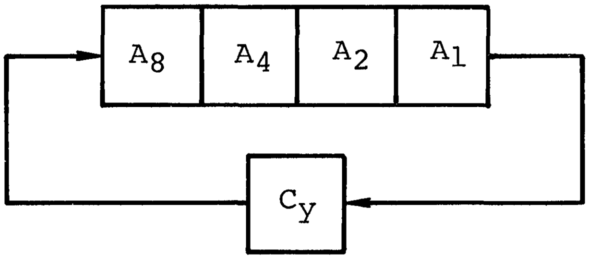
ROTATE RIGHT

RAR

M1				M2			
1	1	1	1	0	1	1	0
8	4	2	1	8	4	2	1

F6

The content of the accumulator and carry are rotated right 1 bit position.



TRANSMIT CARRY AND CLEAR

TCC

M1				M2			
1	1	1	1	0	1	1	1
8	4	2	1	8	4	2	1

F7

The accumulator is cleared. The least significant position of the accumulator is set to the value of the carry. The carry is set to 0. This instruction is used for decimal arithmetic.

DECREMENT ACCUMULATOR

DAC

M1				M2			
1	1	1	1	1	0	0	0
8	4	2	1	8	4	2	1

F8

Decrementing when the accumulator equals zero sets the carry to 0. Decrementing when the accumulator is not zero sets the carry to 1. The initial value of the carry bit does not affect the content of the accumulator.

TRANSFER CARRY SUBTRACT

TCS

M1				M2			
1	1	1	1	1	0	0	1
8	4	2	1	8	4	2	1

F9

The accumulator is set to 9 if the carry is 0. The accumulator is set to 10 if the carry is a 1. The carry is set to 0. This instruction is used for decimal arithmetic.

SET CARRY

STC

M1				M2			
1	1	1	1	1	0	1	0
8	4	2	1	8	4	2	1

FA

Set carry to a 1.

The accumulator is not affected.

DECIMAL ADJUST ACCUMULATOR

DAA

M1				M2			
1	1	1	1	1	1	0	1
8	4	2	1	8	4	2	1

FB

The accumulator is incremented by 6 if either the carry is 1 or if the accumulator content is greater than 9. The carry is set to a 1 if the result generates a carry, otherwise it is unaffected. This instruction is used for decimal arithmetic.

KEYBOARD PROCESS

KBP

M1				M2			
1	1	1	1	1	1	0	0
8	4	2	1	8	4	2	1

FC

A code conversion is performed on the accumulator content, from 1 out of n to binary code. If the accumulator content has more than 1 bit on, the accumulator will be set to 15 (to indicate error). The carry is not affected. The conversion table is shown below:

(ACC) before KBP			(ACC) after KBP	
Hex	Binary		Binary	Hex
0	0 0 0 0	→	0 0 0 0	0
1	0 0 0 1	→	0 0 0 1	1
2	0 0 1 0	→	0 0 1 0	2
4	0 1 0 0	→	0 0 1 1	3
8	1 0 0 0	→	0 1 0 0	4
3	0 0 1 1	→	1 1 1 1	F
5	0 1 0 1	→	1 1 1 1	F
6	0 1 1 0	→	1 1 1 1	F
7	0 1 1 1	→	1 1 1 1	F
9	1 0 0 1	→	1 1 1 1	F
A	1 0 1 0	→	1 1 1 1	F
B	1 0 1 1	→	1 1 1 1	F
C	1 1 0 0	→	1 1 1 1	F
D	1 1 0 1	→	1 1 1 1	F
E	1 1 1 0	→	1 1 1 1	F
F	1 1 1 1	→	1 1 1 1	F

DESIGNATE COMMAND LINE DCL

FD

M1				M2			
1	1	1	1	1	1	0	1
8	4	2	1	8	4	2	1

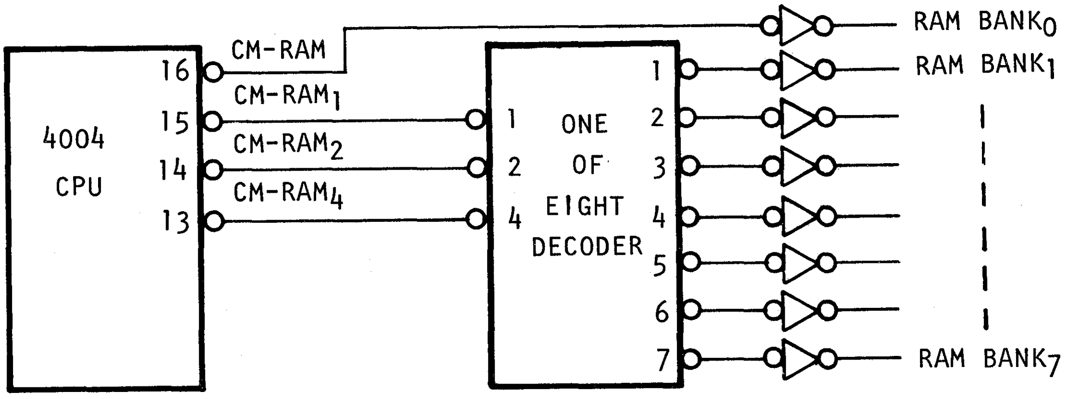
The content of the three least significant accumulator bits is transferred to the CM-RAM output lines on the CPU.

This instruction provides RAM bank selection when multiple RAM banks are used. When the CPU is reset, RAM Bank zero is automatically selected. DCL remains latched until it is changed or reset.

The selection is made according to the following truth table:

(ACC) 8421	CM-RAM$_i$ Enabled	Bank No.
X000	CM-RAM	Bank 0
X001	CM-RAM$_1$	Bank 1
X010	CM-RAM$_2$	Bank 2
X100	CM-RAM$_4$	Bank 3
X011	CM-RAM$_1$, CM-RAM$_2$	Bank 4
X101	CM-RAM$_1$, CM-RAM$_4$	Bank 5
X110	CM-RAM$_2$, CM-RAM$_4$	Bank 6
X111	CM-RAM$_1$, CM-RAM$_2$, CM-RAM$_4$	Bank 7

A low power TTL one-of-eight decoder may be tied to the CM-RAM$_1$, CM-RAM$_2$, and CM-RAM$_4$ lines to expand the number of RAM banks to 8. The command lines must be buffered for MOS compatibility.



The DCL instructions does not apply to PLS 401 and PLS 402 systems since they have capacity for only 4 RAM devices wired to RAM Bank 0.

7. IMPLEMENTING PROGRAMMED LOGIC

Logic diagrams using graphic symbology are the key to visualization and implementation of hardwired logic designs. The sequential nature of programmed logic does not lend itself to logic diagrams. The visual and verbal aids available to the program logic designer are block diagrams, flow charts, register maps, and coding forms.

The designer begins with a block diagram to make input and output, ROM, and RAM register assignments. The problem is flow charted and detailed assignments made of registers on register maps. The flow charts are progressively partitioned into more and more detail until each flow symbol can be converted to program instructions on the coding forms.

The instructions are first written in mnemonic form for easily verbalizing the solution to the problem. When the complete problem or a workable partition has been solved, the mnemonic instructions are converted to code for the ROM. The code is programmed into the ROM and tested with the hardware on the breadboard system.

SYSTEM BLOCK DIAGRAM

One of the first steps in implementing programmed logic is construction of a system block diagram showing assignment of the external input and output connections. Figure 7-1 represents a block diagram work sheet available for PLS-401 systems. Similar worksheets for other PLS-400 systems are available. The worksheet block diagrams show the elements of the PLS systems, and the lines in and out of the system that are available to the designer. Figure 7-1 shows the number of inputs and outputs, RAM memory capacity, ROM program memory capacity, and the CPU microprocessor, interconnected via the CPU bus. The program sequence stored in ROM program memory controls the interaction between the elements connected on the CPU bus.

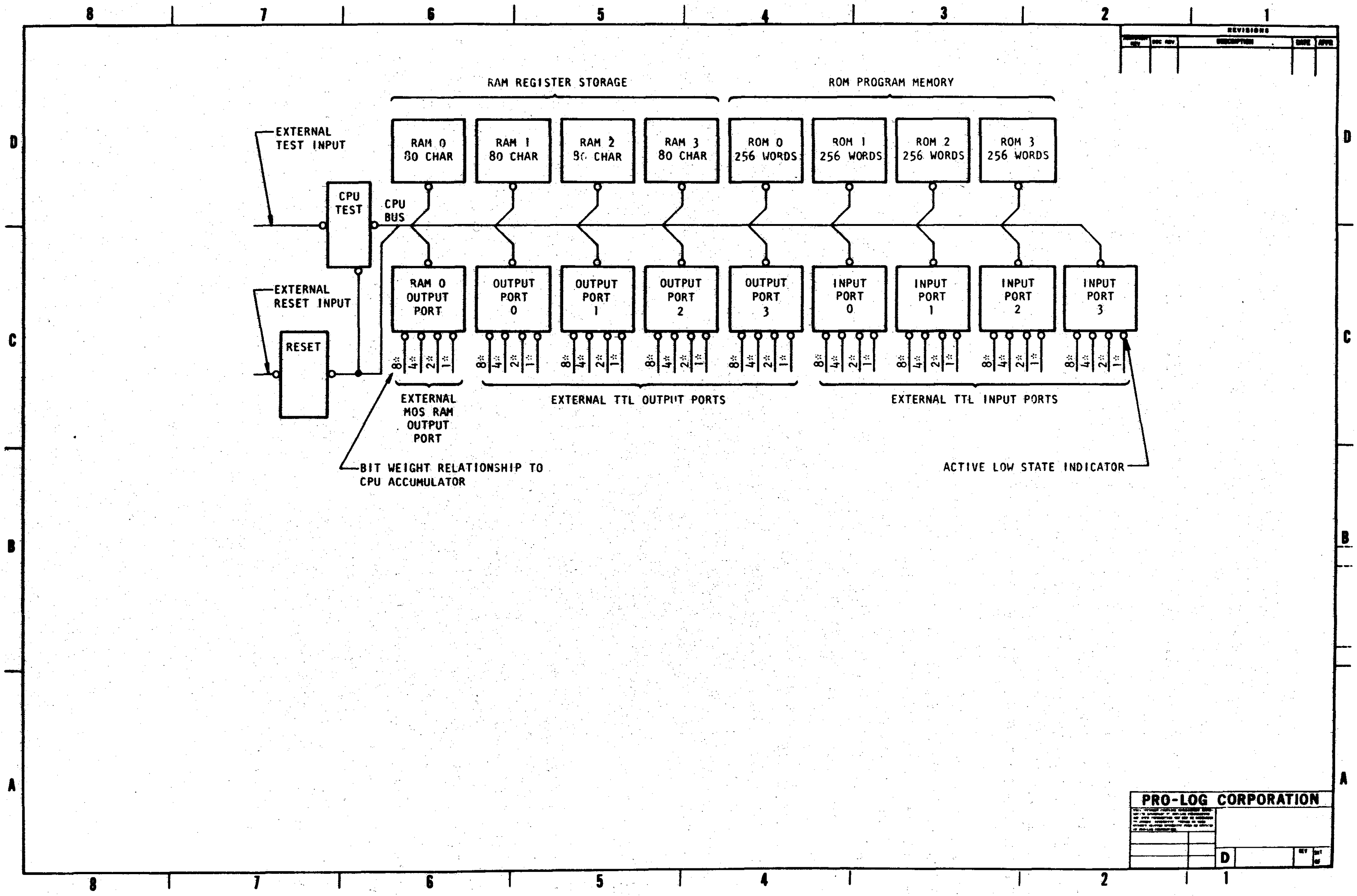


FIGURE 7-1

System Block Diagram

FLOW CHARTS

The sequential nature of programmed logic fits directly into the graphic representation provided by flow charts. Programmed logic being sequential with only yes or no decisions allows for very simple flow charting procedures The graphic symbols used in examples in this manual are shown in Figure 7-2. The main symbols being the rectangle for operations and the diamond for decisions. The rectangle contains an abbreviated statement of the operation or operations. The diamond contains an abbreviated question concerning the decision.

The PLS 400 systems have only two decision instructions, JCN and ISZ. Any time a diamond symbol appears in the flow chart one of these two instructions must be involved. All other instructions perform operations, either alone or in groups, and are thus represented by the rectangle. The use of flow charts correlates to the use of logic block diagrams in hardware design. The hardware designer progressively partitions his problem into more and more detailed logic diagrams until each block represents a logic device. The program logic designer uses progressively more detailed flow diagrams to the point where individual instructions or groups of instructions can be written for each flow symbol. For examples of flow charts refer to Section 8 of this manual.

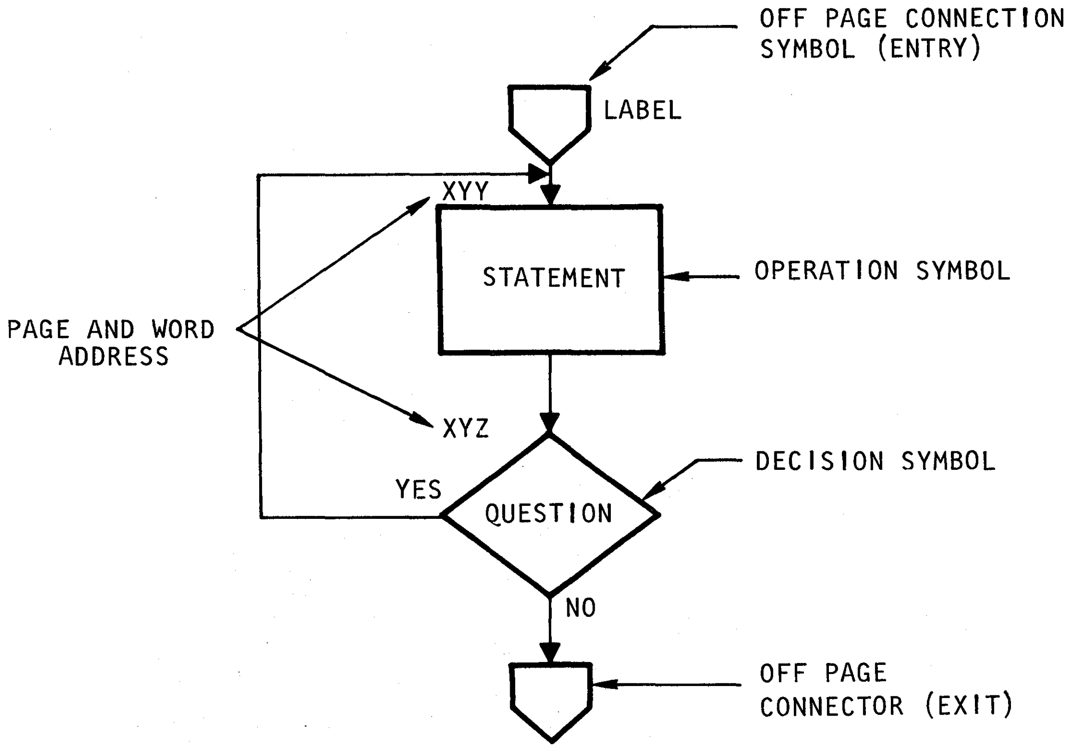


FIGURE 7-2

Flow Chart

REGISTER MAPS

In addition to the block diagram and flow charts, register maps are valuable for visualizing register storage allocation. Figure 7-3 illustrates maps for both the index registers and the RAM registers.

The index register map shows the 16 four bit registers organized such that they can be referred to either individually or in pairs as used by the CPU instructions. The RAM register map shows all the bits available in one 4002 device. The organization is four registers of 20 four bit characters addressable by SRC. Each register contains 16 characters addressable by SRC and 4 characters addressable by individual instructions.

When using register maps it is helpful to write an abbreviated mnemonic on the map to verbalize its assignment.

A mnemonic is written for each register used in a routine to show which registers have been used and what they are used for. When a register is used for individual flag bits it is helpful to expand these in a table showing the individual bit assignment.

A convenient place for recording register maps is on the document containing the flow chart. Examples of register mapping are given in Section 8 of this manual.

INDEX REGISTER MAP

EVEN	8 4 2 1	REG PAIR	8 4 2 1	ODD
E		P7		F
C		P6		D
A		P5		B
8		P4		9
6		P3		7
4		P2		5
2		P1		3
0		P0		1

RAM REGISTER MAP

	REG CHAR																STATUS CHAR						
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	D	1	2	3			
																					8	REG 0	
																					4		
																					2		
																					1		
																					8	REG 1	
																					4		
																					2		
																					1		
																					8	REG 2	
																					4		
																					2		
																					1		
																					8	REG 3	SRC EVEN
																					4		
																					2		
																					1		

FIGURE 7-3
Register Maps

HEX CODING FORM

The Hex Coding form, or some variation of the form, is an absolute necessity for keeping track of the bookeeping details inherrent in programming. In addition, when properly implemented, the coding form becomes the program listing defining how the program accomplishes its task. The program listing in programmed logic is equivalent to a combined logic schematic, wire list and assembly drawing of a hard-wired logic system.

The hex coding form is divided into two major sections each serving a distinct requirement. The major portion of the form is used for mnemonic listing of the program as it is generated. The two left most columns constitute the second section which provides the hexadecimal coding of addresses and instructions used by the CPU and Program memory.

The mnemonic section of the form is completed first as the designer sequentially lists the program steps in easy to remember mnemonic form. When the designer has solidified the mnemonic listing the hex address and instruction codes are assigned. The coding operation of programmed logic is similar to assigning device location, pin numbers, and wire listing in hardware logic.

The mnemonic listing of instructions in programmed logic is equivalent to the hardware logic operation of creating a logic schematic diagram where the program designer assembles instructions and subroutines in a mnemonic list and the hardware designer selects gates MSI and LSI.

① Hexadecimal program memory page address. Single character for 16 pages of program memory.

② Hexadecimal program memory word address. Two hexadecimal character for 256 words per page of memory with the least significant hex digit preprinted on the form.

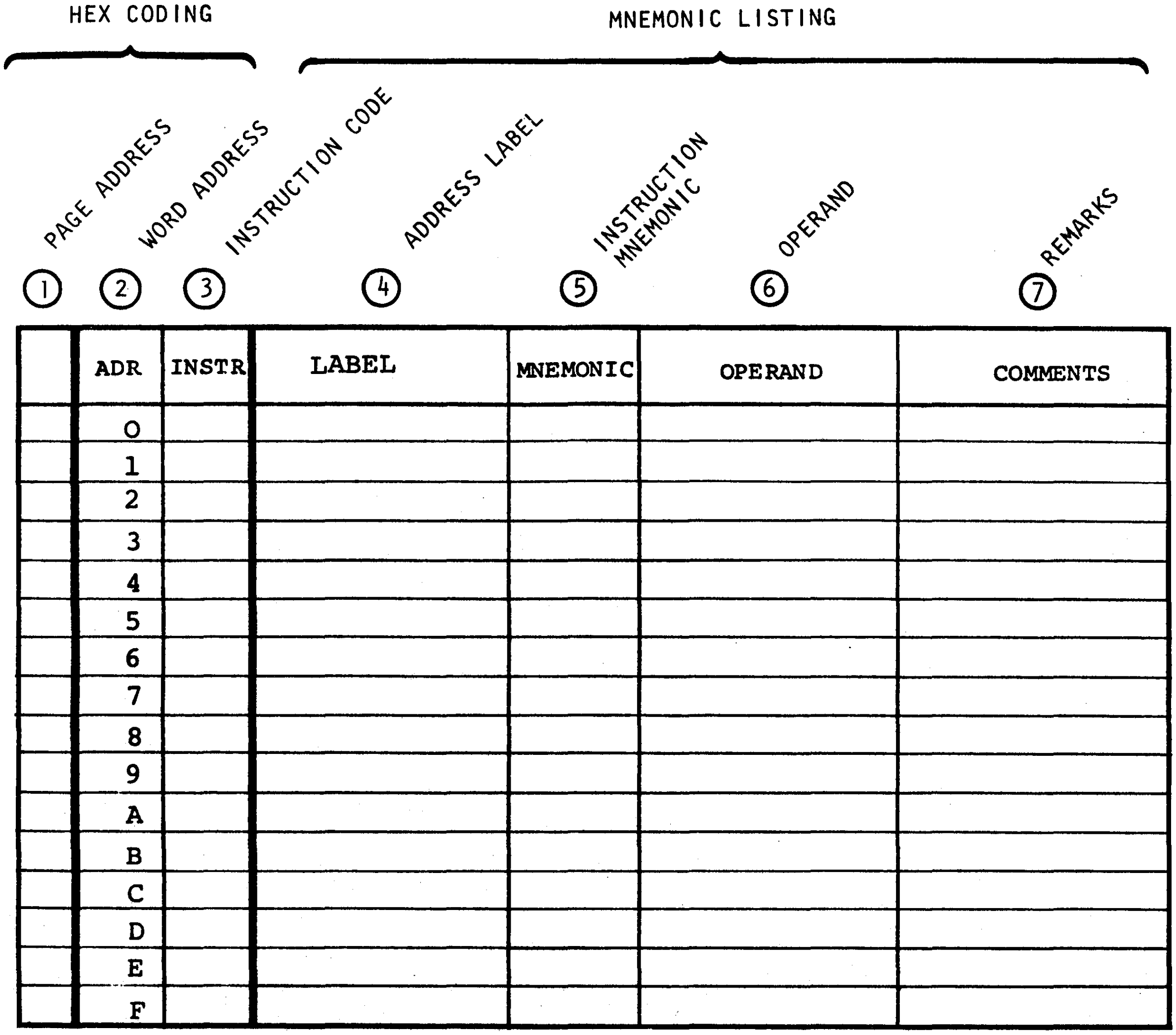
③ Hexadecimal instruction word, as cross referenced between mnemonic and hex code from Section 5.

④ Mnemonic address label used to verbalized the destination of the address control instructions. Address labels in this column must appear only once with each label having unique spelling. This column is left blank if the line does not require a label.

⑤ Mnemonic instructions, usually an abbreviation that verbalizes the operation. The second word of double word instructions does not have a mnemonic and is left blank. The exception is the FIM where the even register data character is inserted. See Figure 7-5.

(6) Mnemonic operand which can be blank, data constants, instruction modifiers, register designation, register pair designation, or a source address label. Instructions 0 through 9 and A through D always require operand information. I/O, RAM and accumulator instructions never have operand information. See Figure 7-5. for examples of operands.

(7) Written comments defining the purpose of an instruction or a group of instructions.



	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	A					
	B					
	C					
	D					
	E					
	F					

FIGURE 7-4

Hex Coding Form

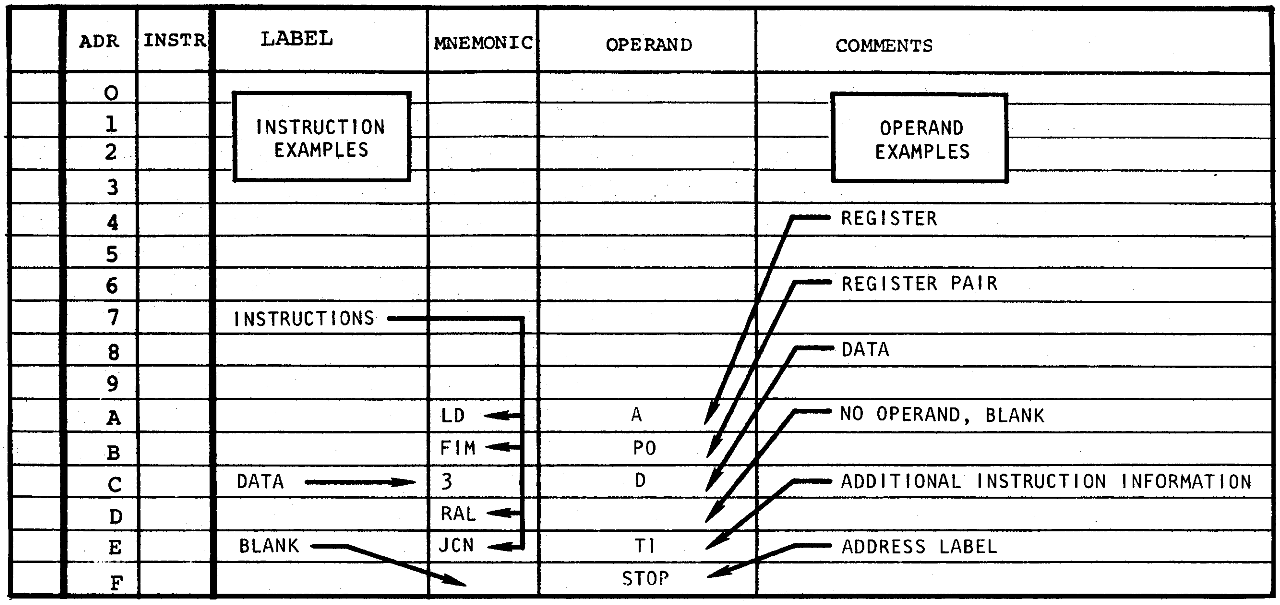


FIGURE 7-5

Instruction and Operand Examples

PLS DESIGN EXAMPLE

An example for implementing an electronic calculator is given to illustrate the techniques of using the block diagram, flow charts, register maps, and coding forms. The problem is defined as being a four function eight digit calculator where two separate entries of up to eight digits are operated upon. Each entry is displayed as it is keyed into the unit. The result of the operation is displayed following entry of the second operation.

The block diagram is generated as shown in Figure 7-6, consisting of a PLS 401 system, eight digits of latching display, and a 16 key keyboard. The keyboard is assigned as a four-by-four matrix, with four key columns driven by four output lines and the key closures sensed as four input row lines.

The latching displays are connected for separate strobe inputs and common BCD data lines plus decimal point. The display data lines are shared with the keyboard matrix column selection lines. An additional key is used on the external reset input for a clear function.

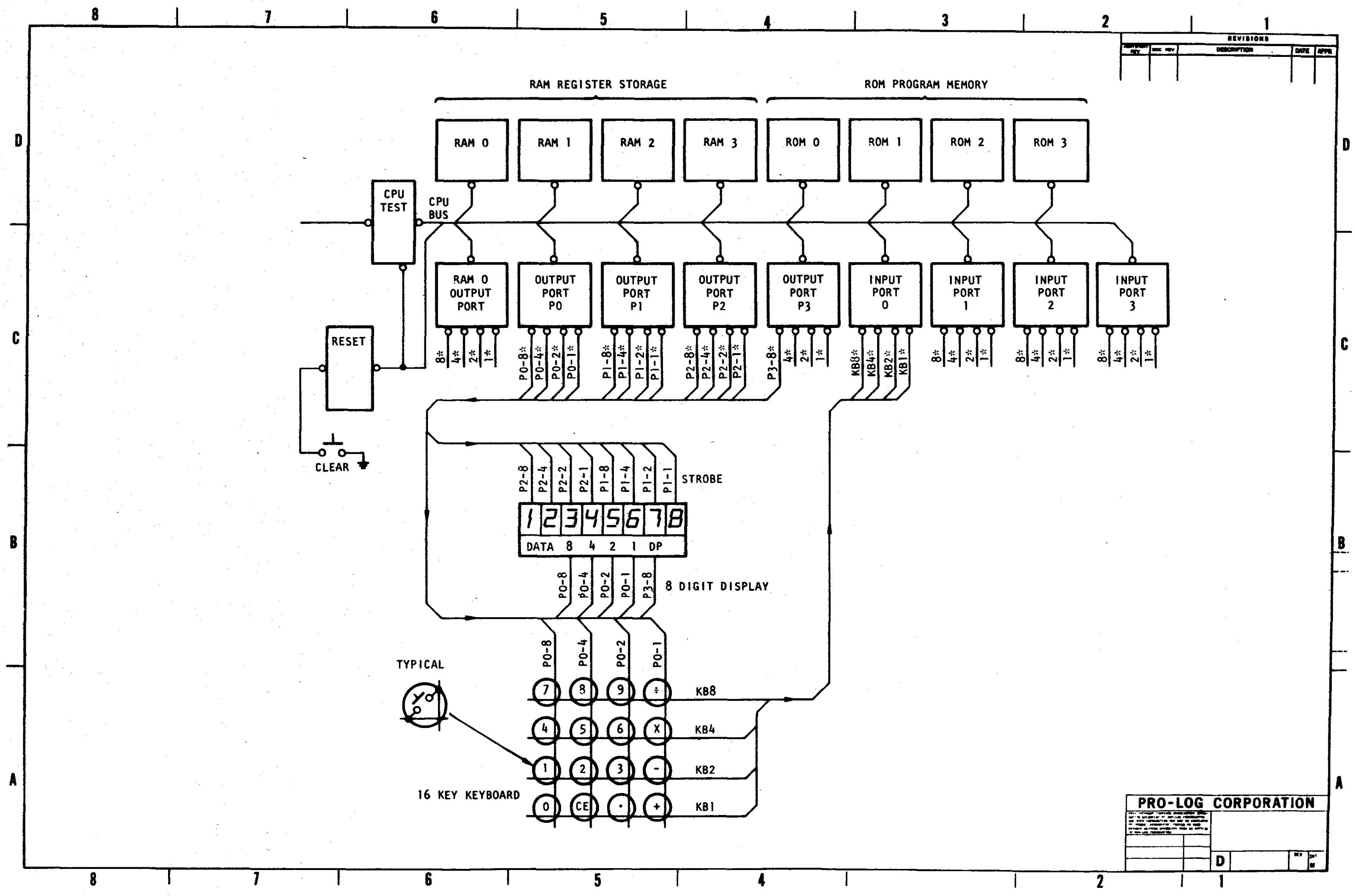
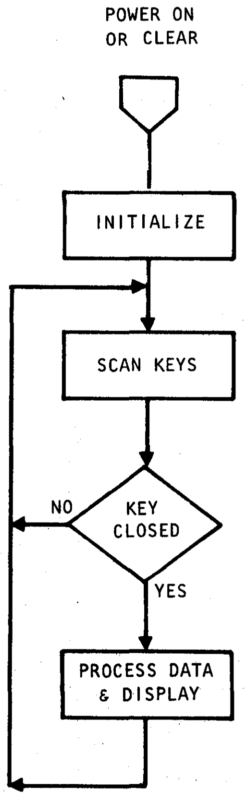


FIGURE 7-6

Electronic Calculator Block Diagram

A very basic flow diagram, Figure 7-7, is generated which shows the two major operations of scanning the keys and processing the data. In addition, a RAM register map is generated showing assignment of the entries to be processed.



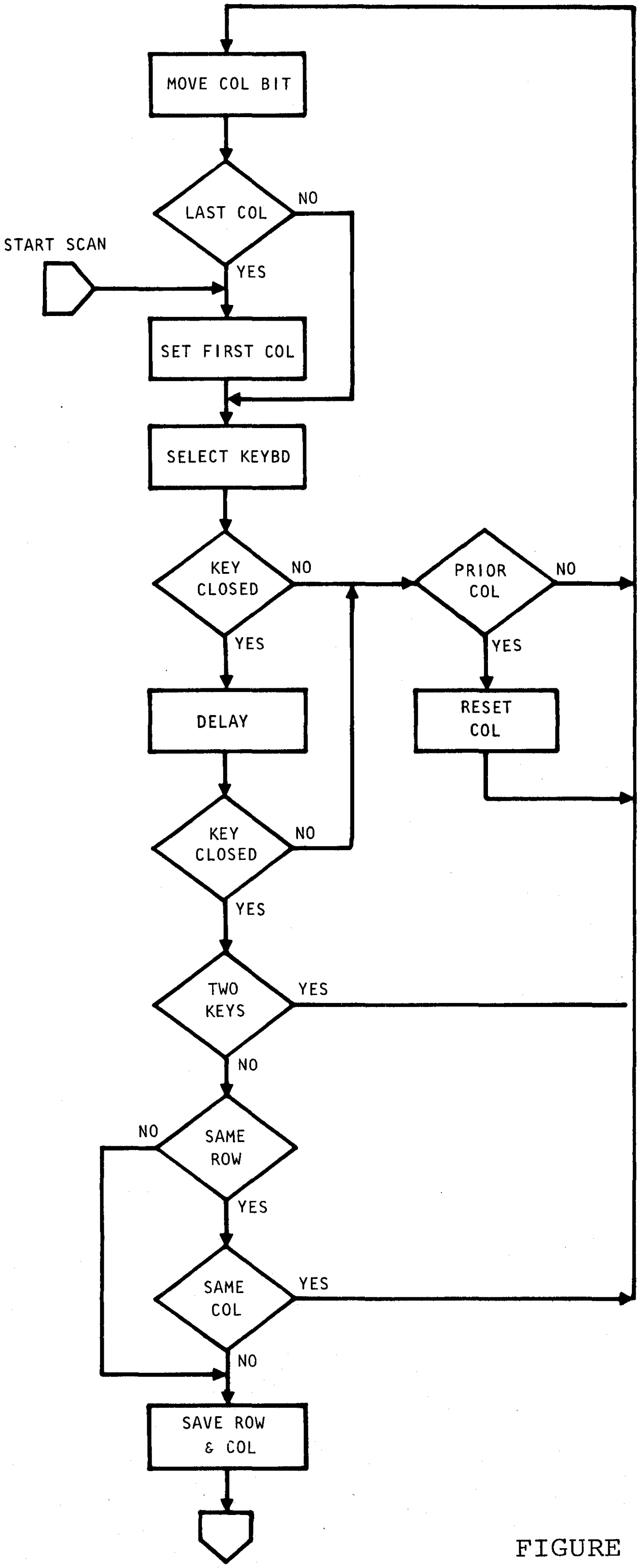
REG 0	FIRST ENTRY OR RESULT				
REG 1	SECOND ENTRY				
REG 2					
REG 3					

RAM REGISTER MAP

FIGURE 7-7

First Level Flow Chart

The operation of scanning the keys is expanded upon as shown in Figure 7-8. The main purpose of this routine is to scan the keyboard matrix for a closure and to debounce the asynchronous key closures and openings. At this time, three registers are assigned on the index register map, KEY ROW, KEY COL and COL COUNT. KEY ROW and KEY COL are used to store the row and column bits of a detected key. COL COUNT is used to keep track of which column is being scanned.



EVEN		PAIR		ODD
E	KEYROW	7	KEYCOL	F
C		6	COL COUNT	D
A		5		B
8		4		9
6		3		7
4		2		5
2		1		3
0		0		1

INDEX REGISTER MAP

FIGURE 7-8

Flow Chart for Keyboard Scan

The four key columns are scanned one at a time by rotating a bit through the columns. As each column is scanned the rows are read as inputs and tested for a key closure. If a key closure is detected a debouncing delay is generated and the same column is read again. If the key is still closed, it is checked to determine if the same key had previously been closed. If the closure was the same key detected in a previous scan, the routine ignores the key and returns to scanning the key columns. If the key was not previously closed, the row and column of the new key are stored and the routine exits to process the key.

If no key closure is detected on a column, the column register is tested to see if a key in that column was previously closed. If it is the same column where a key was previously closed, the column register is reset indicating that a key was just released. The routine returns to column scanning. If the key that was just released bounces and is detected on the next scan, the debounce delay and second read should find the key open.

The operation of processing keys from the keyboard is expanded in the flow chart of Figure 7-9. The purpose of this routine is to decide the key matrix so the indicated function may be performed.

The KEY COL register is examined to determine if the key closure occurred in Column 1. If the closure was in Column 1 the individual bits of KEY ROW are examined to determine which of the function keys (+, -, X, or ÷) was closed. If the closure was not in Column 1, ROW 1 is examined to determine if either of the other functions CE or CP are closed.

If the closure is determined to be a data key, the row and column data is converted to a single hex character and used as an address for a lookup table. The table translates the key matrix address to the appropriate decimal key data.

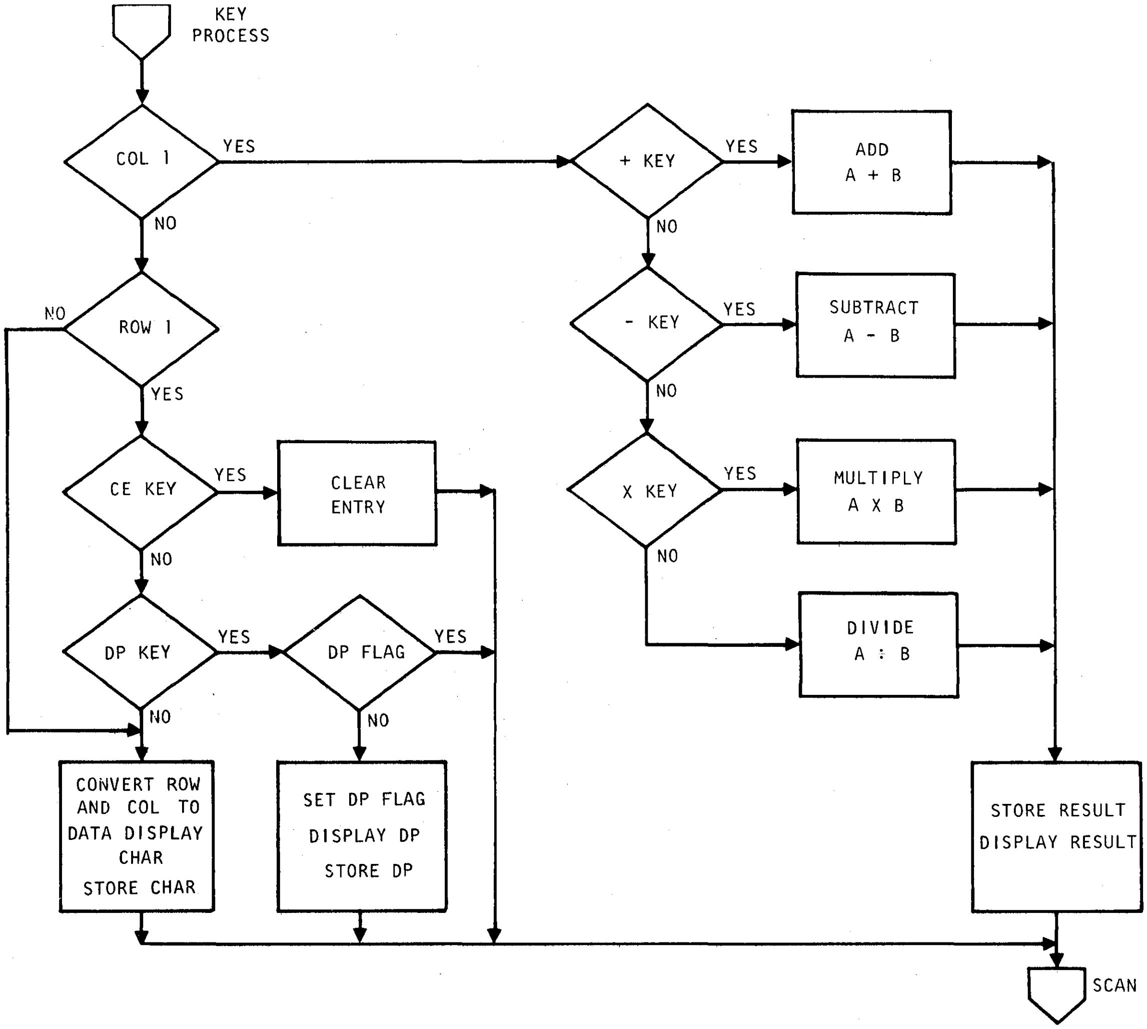


FIGURE 7-9

Flow Chart for Key Process

Mnemonic Listing for Keyboard Scan

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0		SCAN	LD	D	CHECK FOR LAST COL
	1			RAL		
	2			JCN	CO	
	3				NOT LAST	
	4		START SCAN	LDM	1	SET COL COUNT FOR COL 1
	5			XCH	D	
	6		NOT LAST	FIM	PO	SELECT KEY COL
	7			O	O	
	8			SRC	PO	
	9			LD	D	
	A			WRR		
	B			RDR		READ KEYBOARD
	C			JCN	AO	
	D				NO KEY	
	E			FIM	PO	DEBOUNCE KEY
	F			F	C	
	0		△1	ISZ	O	
	1				△1	
	2			ISZ	1	
	3				△1	
	4			RDR		READ KEYBOARD
	5			JCN	AO	
	6				NO KEY	
	7			KBP		CHECK FOR DOUBLE KEY
	8			CMA		
	9			JCN	AO	
	A				SCAN	
	B			CLC		SAVE NEW ROW DATA
	C			CMA		
	D			XCH	E	
	E			SUB	E	CHECK IF SAME ROW.
	F			JCN	A1	
	0				SAVE COL	
	1			CLC		CHECK IF SAME COL
	2			LD	D	
	3			SUB	F	
	4			JCN	AO	
	5				SCAN	
	6		SAVE COL	LD	D	SAVE NEW COL
	7			XCH	F	
	8			—		PROCESS NEW KEY
	9			—		
	A			—		
	B			—		
	C					
	D		NO KEY	CLC		CHECK IF SAME COL
	E			LD	D	
	F			SUB	F	
	0			JCN	A1	
	1				SCAN	
	2			XCH	F	RESET KEY COL
	3			JUN		
	4				SCAN	

Mnemonic Listing for Keyboard Process

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0		KEY PROCESS	LD	F	⊤ CHECK IF KEY IS IN COL 1
	1			RAL		(+, −, X, ÷)
	2			JCN	CO	
	3				NOT COL 1	↓
	4			LD	E	⊤ CHECK IF ADD KEY
	5			RAR		
	6			JCN	C1	
	7				PLUS	↓
	8			RAR		⊤ CHECK IF SUB KEY
	9			JCN	C1	
	A				MINUS	↓
	B			RAR		⊤ CHECK IF MULT KEY
	C			JCN	C1	
	D				TIMES	↓
	E			—		⊤ DIVIDE A by B
	F			—		
	0			—		
	1			—		
	2			—		↓
	3		NOT COL 1	LD	E	⊤ CHECK IF KEY IS IN ROW 1
	4			RAR		
	5			JCN	CO	
	6				NOT ROW 1	↓
	7			LD	F	⊤ CHECK IF CLEAR ENTRY KEY
	8			RAL		
	9			RAL		
	A			JCN	C1	
	B				CLEAR ENTRY	↓
	C			RAL		⊤ CHECK IF DECIMAL POINT KEY
	D			JCN	C1	
	E				DP	↓
	F		NOTROW 1	LD	E	⊤ CONVERT ROW & COL TO A
	0			KBP		HEX DIGIT FOR LOOKUP TABLE
	1			DAC		
	2			XCH	1	
	3			LD	F	
	4			KBP		
	5			DAC		
	6			RAL		
	7			RAL		
	8			CLC		
	9			ADD	1	
	A			XCH	1	↓
	B			LDM	F	⊤ USE HEX DIGIT AS ADDRESS IN
	C			XCH	O	LOCATION FX
	D			FIN	PO	↓
	E			—		⊤ DISPLAY DECIMAL CHAR & STORE
	F			—		
	0			—		
	1			—		↓

Coding for Keyboard Lookup Table

	F0					
	F1					
	F2					
	F3					
	F4					
	F5	03				3 ⊤ KEY BOARD MATRIX TRANSLATION
	F6	06				6
	F7	09				9
	F8					
	F9	02				2
	FA	05				5
	FB	08				8
	FC	00				0
	FD	01				1
	FE	04				4
	FF	07				7 ↓

When the complete problem has been charted and listed in mnemonic form the hexadecimal address and instruction codes are assigned on the coding form. The address and instruction data is then transferred to the program memory for system and program debugging.

8. PROGRAMMING APPLICATIONS

SUBROUTINES

A group of instructions written to perform a function with common usage is referred to as a subroutine. The PLS 400 systems have two instructions, JMS and BBL, which allow subroutines to be used conveniently and efficiently. The JMS instruction allows the program to exit and perform a common routine and the BBL instruction at the end of a subroutine causes the sequence to return to the main program.

As shown in Figure 8-1 the JMS instruction can be used as many times as needed to execute the same subroutine and automatically return to the proper place in sequence in the main program. Using subroutines is very efficient in terms of program storage space if the subroutine is long enough and used often enough. If a subroutine is too short or is not used enough it is possible to waste program storage space. This results because it requires two program locations to enter a subroutine plus one to return in addition to the routine itself. Table 8-1 presents the program locations that can be gained or lost by using subroutines based on how many steps in the routine and how many times the routine occurs. As an example a subroutine of two steps will always result in a loss of three locations no matter how many times it is used. A subroutine of three steps must be used four times to break even.

Subroutines are implemented so that the CPU hardware keeps track of the return address by storing the program address counter in the subroutine address stack when a JMS is executed and by retrieving it back to the program address counter when a BBL is executed.

Nesting

The subroutine address stack in the 4004 CPU can store up to three subroutine return addresses. This feature allows nesting of subroutines as shown in Figure 8-2. Nesting means that a subroutine may have other subroutines within itself. A one-level subroutine may have other subroutines within itself. A one-level subroutine is one which does not contain any other subroutine. A two level-subroutine contains at least one, one-level subroutine

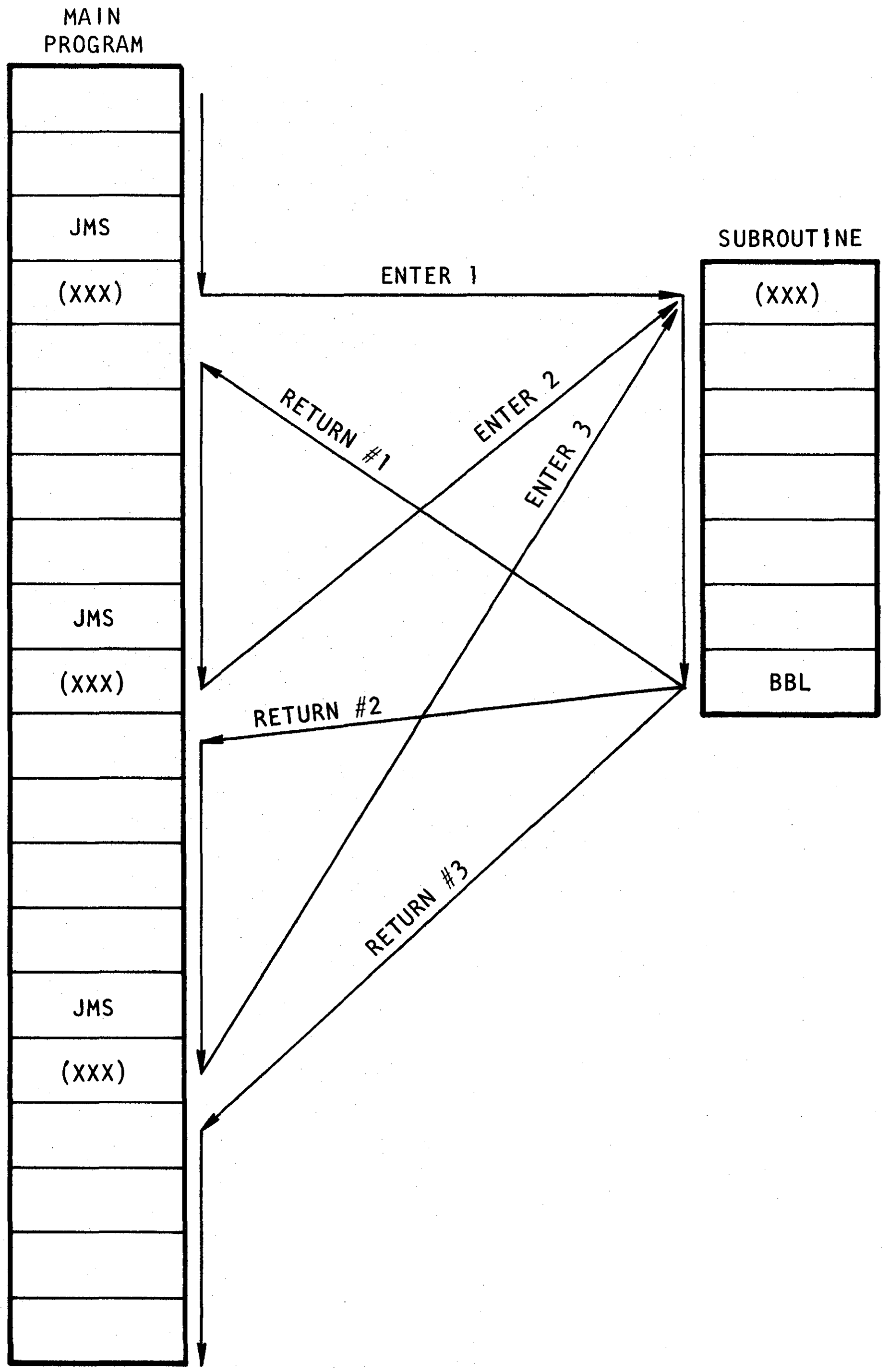


FIGURE 8-1

Example Showing how a Subroutine can be Used Many Times From Various Places in a Routine

TABLE 8-1

Number of Steps Gained or Lost When a Routine is Executed as a Subroutine

Number of Times Routine Occurs (X)

Number of Steps in Routine (N)

	1	2	3	4	5	6	7	X
1	-3	-4	-5	-6	-7	-8	-9	-x -2
2	-3	-3	-3	-3	-3	-3	-3	-3
3	-3	-2	-1	0	1	2	3	x -4
4	-3	-1	1	3	5	7	9	2 X -5
5	-3	0	3	6	9	12	15	3 X -6
6	-3	1	5	9	13	17	21	4 X -7
7	-3	2	7	12	17	11	27	5 X -8
8	-3	3	9	15	21	27	33	6 X -9
9	-3	4	11	18	25	32	39	7 X -10
10	-3	5	13	21	29	37	45	8 X -11
N	-3	N-5	2N-7	3N-9	4N-11	5N-13	6N-15	(N-2)X-(N+1)

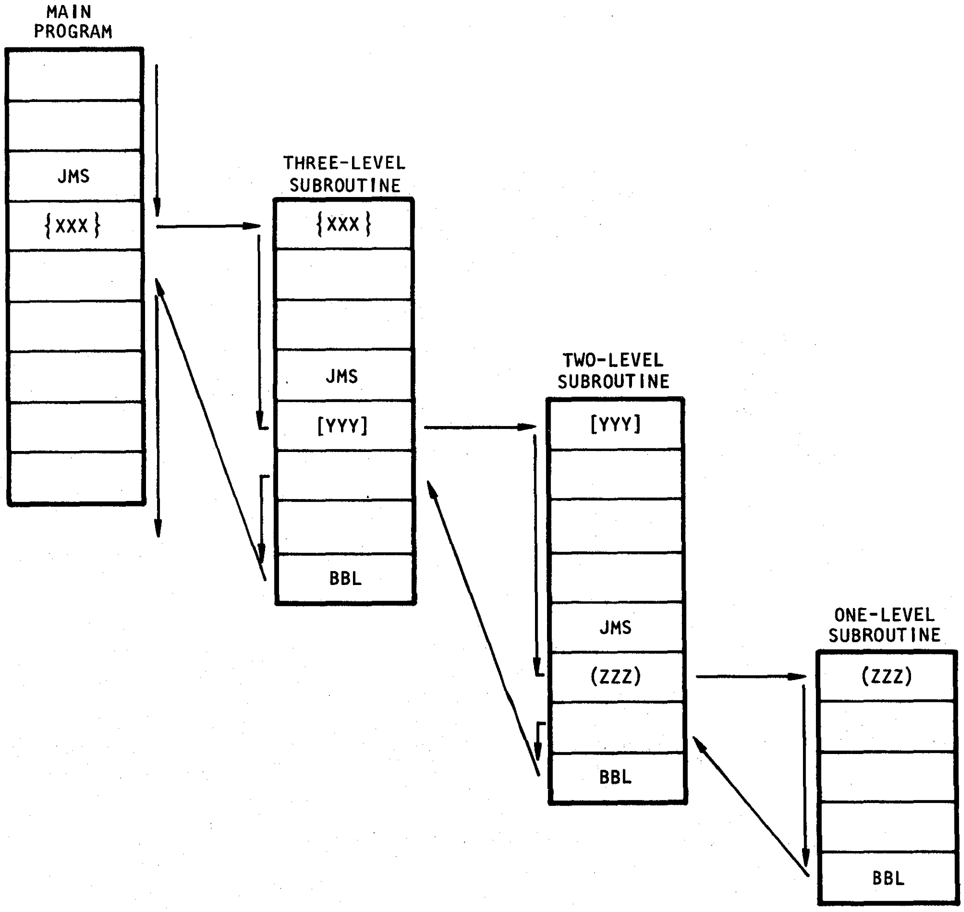


FIGURE 8-2

Example Showing Nesting of Subroutines

and a three-level subroutine contains at least one, two-level subroutine. The subroutine address stack is referred to as a "push-down" stack where each time a new subroutine is entered, the stack is pushed-down with the old return addresses going to the bottom of the stack and the most current being on top. When a BBL return is executed the most current or the address at the top of the stack is used first and the stack is pulled-up one level. The subroutine stack only has three levels and it is possible to push the stack down too far by executing more than three JMS instructions without an intervening return. In order to keep track of the three subroutine levels, parenthesis (), brackets [], and braces {} are used as shown below:

Parentheses, (LABEL X) denotes a one level subroutine.

Brackets, [LABEL Y] denotes a two level subroutine.

Braces, {LABEL Z} denotes a three level subroutine.

The rules for nesting of mathematical factors apply to the nesting of subroutines where any lower level subroutine may be nested within a higher level subroutine. A subroutine like a mathematical factor may not have nested within itself one of its own level or a higher level.

Examples of nesting

1. {[(----)]}

2. {[---] [---] [(---) (---)]}

Multiple Ending Subroutines

The BBL instruction has the feature of forcing a constant value into the accumulator. This feature can be put to use as shown in Figure 8-3 where a subroutine can make decisions and terminate with multiple endings. Each ending can be executed with a BBL with its own constant value forced into the accumulator. Therefore the main program could test the accumulator in order to determine which ending occurred.

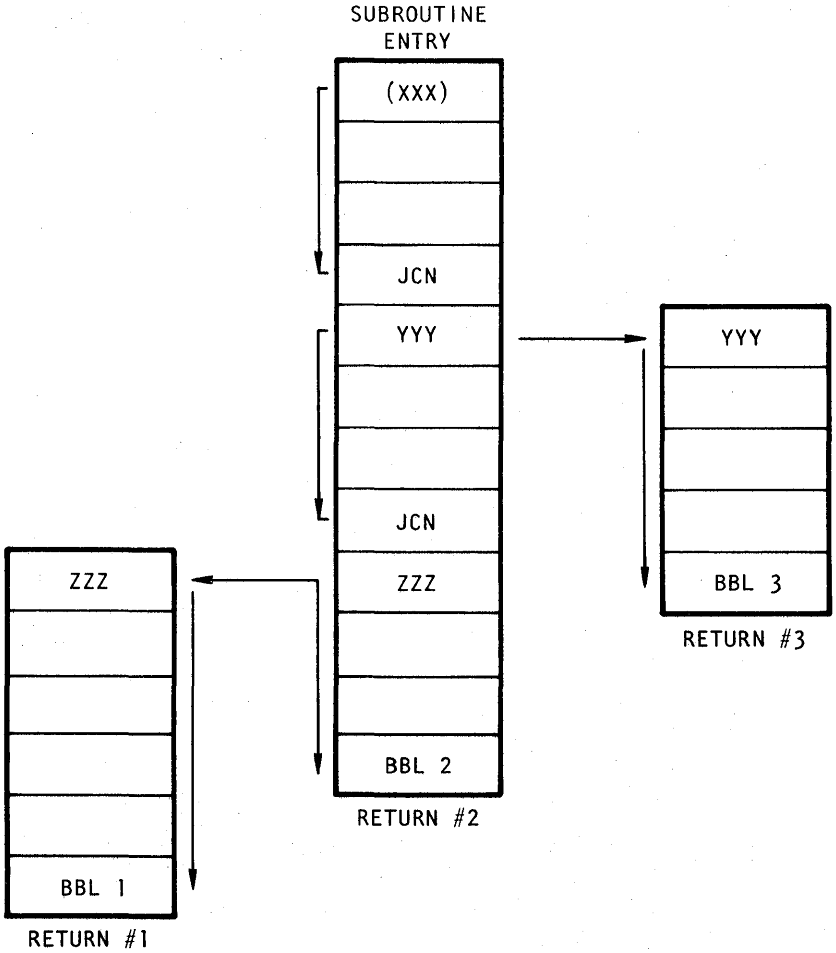


FIGURE 8-3

Example of a Subroutine With Multiple Endings

Common Ending Subroutines

In a complex program it is relatively easy to reach three levels of subroutines. A technique which helps conserve levels is the use of subroutine stacking in specialized situations. Figure 8-4 shows an example where subroutines are stacked to share a common ending. Basically this technique uses a jump unconditional to an existing subroutine rather than pushing the stack down another level. This technique is useful only when a JMS occurs at the very end of a subroutine. When this occurs a JUN is used in place of the JMS. This keeps the stack at the same level.

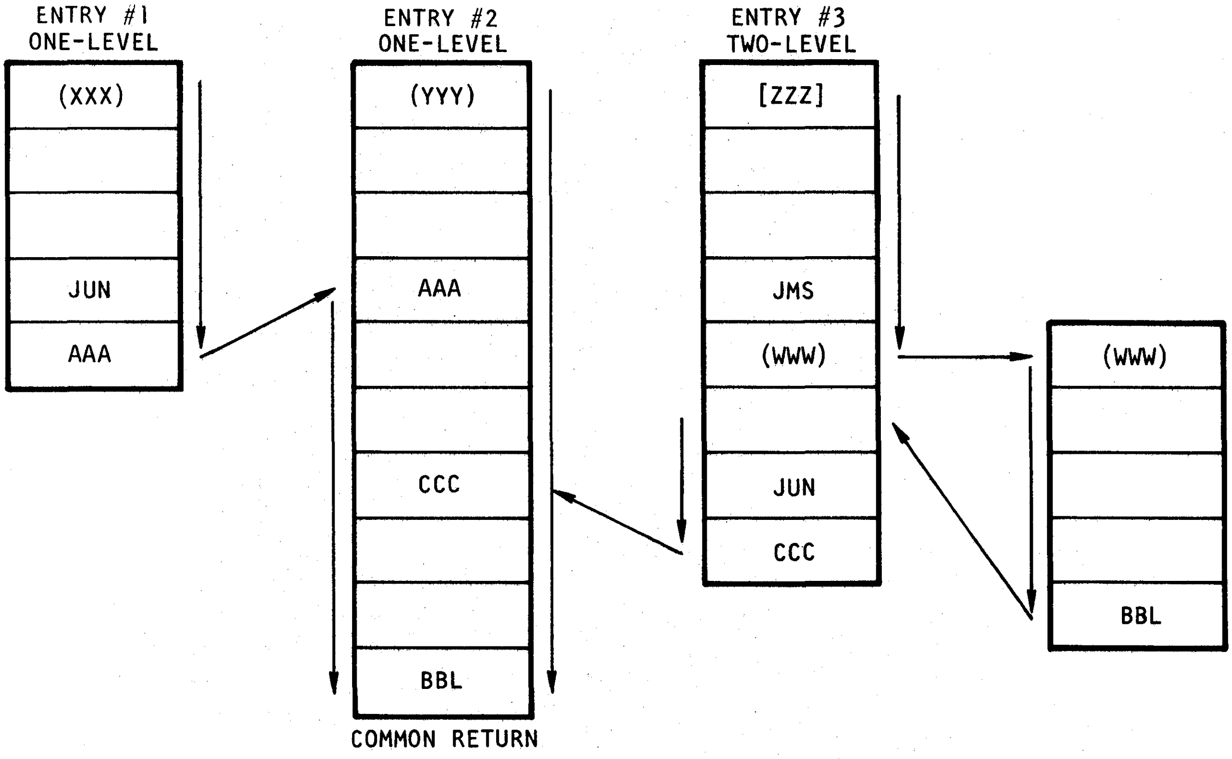


FIGURE 8-4

Example of Subroutines Sharing a Common Ending Sequence (Stacking)

COUNTING

Counting is one of the logical functions PLS systems can easily do with the INC, IAC, DAC, and ISZ instructions. The simplest count is use of INC to perform a binary hex count from 0 through F on any of the index registers. When it is desirable to count greater than 15, the ISZ instructions may be cascaded to reach any practical value. The example below shows cascading ISZ instructions where the first register overflows, the second register is then counted, and when the second register overflows the third register is counted. This technique can be extended to any number of registers for large counts.

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0			—		
	1			—		
	2		COUNT	ISZ	0	INCREMENT FIRST REGISTER
	3				COUNT	
	4			ISZ	1	INCREMENT SECOND REGISTER
	5				COUNT	
	6			INC	2	INCREMENT THIRD REGISTER
	7			—		
	8			—		
	9			—		
	A					

Counters may be used simply to tally up the number of times a function occurs or in other situations they may be used to control the number of times a function occurs. In the control situation, the count is compared to some preset limit and the routine terminated when the limit is reached. The ISZ instruction provides an efficient means for performing this control function by counting preset registers and terminating when the registers overflow.

Two options exist in counting; one is to execute the functions to be counted before counting and the other is to count first and then execute. This distinction is important when presetting registers to be counted. In the execute and count technique the function will always be executed at least once, since the decision to terminate is a function of the count. In the count and execute technique it is possible for the count to terminate the routine without performing any execution. Figures 8-5 and 8-6 show two ways to implement these techniques.

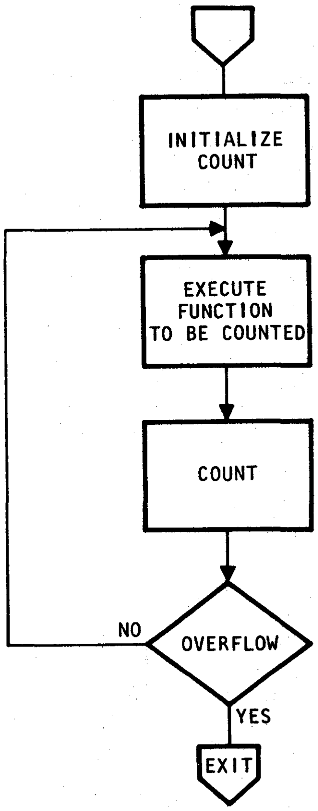


FIGURE 8-5

Flowchart of Execute and Count

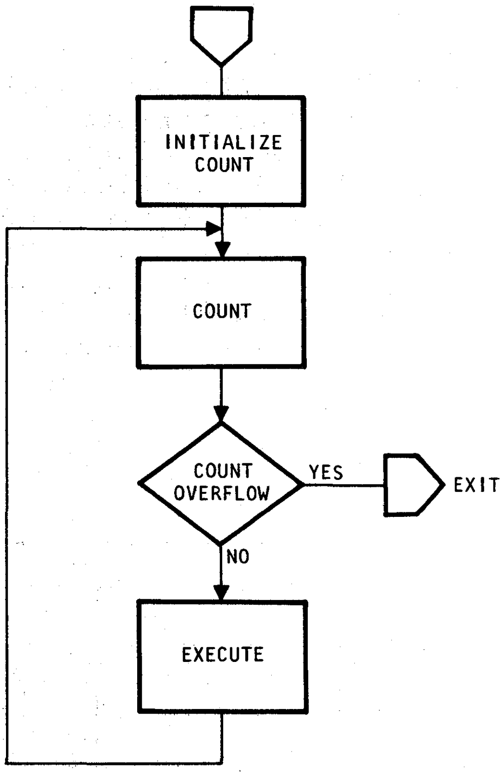


FIGURE 8-6

Flowchart of Count and Execute

Coding for execute and count

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	40			—		
	41			—		
	42	20		FIM	P0	INITIALIZE THE COUNTER
	43	00		0	0	
	44	22		FIM	P1	
	45	00		0	0	
	46	—	EXECUTE	—		EXECUTE SOME FUNCTION
	47	—		—		
	48	—		—		
	49	—		—		
	4A	—		—		
	4B	—		—		
	4C	70		ISZ	0	COUNT HOW MANY EXECUTIONS
	4D	46			EXECUTE	
	4E	71		ISZ	1	
	4F	46			EXECUTE	
	50	72		ISZ	2	
	51	46			EXECUTE	
	52	73		ISZ	3	
	53	46			EXECUTE	
	54	—		—		OVERFLOW OR ENDING ROUTINE
	55	—		—		
	56	—		—		
	57	—		—		
	58	—		—		

Coding for count and execute

ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
0			—		
1			—		
2			—		
63	20		FIM	PO	INITIALIZE THE COUNTER
64	D₀D₁		D₀	D₁	
65	22		FIM	P1	
66	D₂D₃		D₂	D₃	
67	70	COUNT	ISZ	O	COUNT HOW MANY EXECUTIONS
68	74			EXECUTE	
69	71		ISZ	1	
6A	74			EXECUTE	
6B	72		ISZ	2	
6C	74			EXECUTE	
6D	73		ISZ	3	
6E	74			EXECUTE	
6F			—		OVERFLOW ROUTINE
70			—		
71			—		
72					
73					
74		EXECUTE	—		EXECUTE SOME FUNCTION
75			—		
76			—		
77			—		
78			—		
79	1		JCN	A1	
7A	67			COUNT	
7B			—		ENDING ROUTINE
7C			—		
7D			—		

Binary Count

The nature of the ISZ instruction requires that the preset count provide a binary complement limit of the desired count. For the execute/count situation the preset value is the complement plus one and for the count/execute technique the preset value is the straight complement. Table 8-2 is presented as a convenience for determining register settings for counting with cascaded ISZ instructions. Register 0 must be set according to whether the technique is count/execute or execute/count. The total desired count is determined by adding all the individual register counts.

Example for count of 2387 using execute/count:

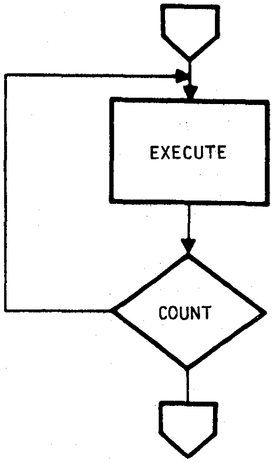
	Setting	N_X
Reg 2	6	2304
Reg 1	A	80
Reg 0	D	3
		2387

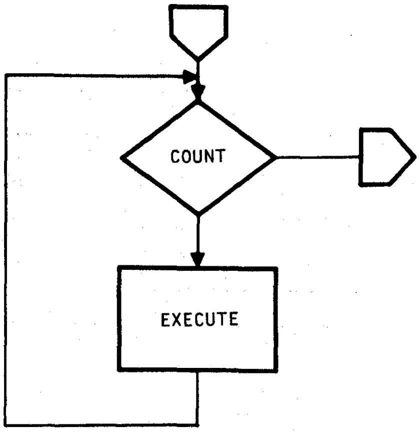
TABLE 8-2

ISZ Register Settings for "N" Operations

REG O EXEC COUNT	REG O COUNT EXEC	REG 1 N1	REG 2 N2	REG 3 N3	REG 4 N4
F 1	F 0	F 0	F 0	F 0	F 0
E 2	E 1	E 16	E 256	E 4096	E 65536
D 3	D 2	D 32	D 512	D 8192	D 131072
C 4	C 3	C 48	C 768	C 12288	C 196608
B 5	B 4	B 64	B 1024	B 16384	B 262144
A 6	A 5	A 80	A 1280	A 20480	A 327680
9 7	9 6	9 96	9 1536	9 24576	9 393216
8 8	8 7	8 112	8 1792	8 28672	8 458752
7 9	7 8	7 128	7 2048	7 32768	7 524288
6 10	6 9	6 144	6 2304	6 36864	6 589824
5 11	5 10	5 160	5 2560	5 40960	5 655360
4 12	4 11	4 176	4 2816	4 45056	4 720896
3 13	3 12	3 192	3 3072	3 49152	4 786432
2 14	2 13	2 208	2 3328	2 53248	2 851968
1 15	1 14	1 224	1 3584	1 57344	1 917504
0 16	0 15	0 240	0 3840	0 61440	0 983040

$$N_{TOTAL} = N_0 + N_1 + N_2 + N_3 + N_4$$

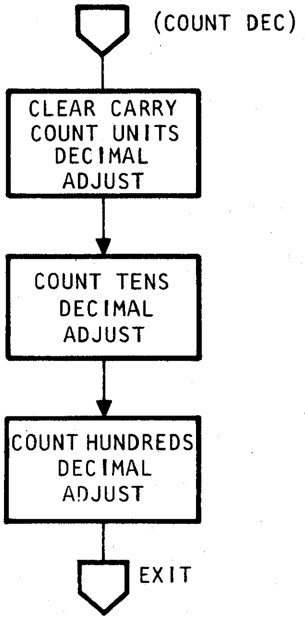




Decimal Count

All CPU instructions count directly in binary. When it is necessary to do a decimal count such as for displays, the DAA instruction becomes useful.

An example of a subroutine to count three decimal decades is given in Figure 8-7. The carry is initially cleared. The units decade is loaded to the accumulator and incremented. The accumulator is decimal adjusted and the result saved as the new units decade. The TCC instruction moves the carry, if any, to the accumulator and the tens decade is added. The accumulator is again decimal adjusted and saved with the TCC moving any decimal overflow to the accumulator for adding the hundreds decade.


(COUNT DEC)
CLEAR CARRY COUNT UNITS DECIMAL ADJUST
COUNT TENS DECIMAL ADJUST
COUNT HUNDREDS DECIMAL ADJUST
EXIT

EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6		3	HUNDREDS	7
4	TENS	2	UNITS	5
2		1		3
0		0		1

INDEX REGISTER MAP

FIGURE 8-7

Subroutine to Count Three Decimal Decades

ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
0		(COUNT DEC)	CLC		INCREMENT UNITS
1			LD	5	
2			IAC		COUNT
3			DAA		DECIMAL ADJUST
4			XCH	5	SAVE
5			TCC		INCREMENT TENS
6			ADD	4	
7			DAA		DECIMAL ADJUST
8			XCH	4	
9			TCC		INCREMENT HUNDREDS
A			ADD	7	
B			DAA		DECIMAL ADJUST
C			XCH	7	
D			BBL	0	RETURN
E					
F					

TIME DELAYS

Time delay circuits can be simulated with programmed logic using simple counting techniques. Since each instruction word requires 10.8 microseconds to execute, the simplest time delay can be achieved by executing a number of nonoperative instructions such as the NOP. Using this method, any significant time delay would soon use up considerable program memory space.

A more efficient time delay can be implemented using the ISZ instruction executed so that it loops on itself.

X	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	20					
	21	70	SELF	ISZ	0	TIME DELAY
	22	21			SELF	
	23					

For a 10.8 microsecond instruction cycle each execution of the two word ISZ instruction uses 21.6 microseconds. Therefore, if an ISZ instruction initially starts from 0 and loops through 16 passes before it leaves the loop, a total of 345.6 microseconds is required.

By cascading two ISZ instructions one after another with the loops of both returning to the first ISZ, the time delay will be doubled for each pass through the second ISZ.

X	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	20					
	21	70	FIRST	ISZ	0	CASCADED TIME DELAY
	22	21			FIRST	
	23	71		ISZ	1	
	24	21			FIRST	

If two registers, started from zero, are cascaded in this fashion a total of 5.88 milliseconds is required before the program exits the loop. Cascading of additional registers will further increase the time delay in an exponential manner.

Table 8-3 is presented for determining the register settings for ISZ cascading up to six registers.

TABLE 8-3

Delay Time using Cascaded ISZ Instructions

N	REG 0	T0 MICRO SEC	REG 1	T1 MILLI SEC	REG 2	T2 MILLI SEC	REG 3	T3 SEC	REG 4	T4 SEC	REG 5	T5 SEC
1	F	21.6	F	0.022	F	0.02	F	0.000	F	0.00	F	0.0
2	E	43.2	E	0.389	E	5.92	E	0.094	E	1.51	E	24.2
3	D	64.8	D	0.756	D	11.82	D	0.189	D	3.02	D	48.3
4	C	86.4	C	1.123	C	17.71	C	0.283	C	4.53	C	72.4
5	B	108.0	B	1.490	B	23.61	B	0.378	B	6.04	B	96.6
6	A	129.6	A	1.858	A	29.51	A	0.472	A	7.55	A	120.7
7	9	151.2	9	2.225	9	35.40	9	0.566	9	9.06	9	144.9
8	8	172.8	8	2.592	8	41.30	8	0.660	8	10.56	8	169.0
9	7	194.4	7	2.959	7	47.20	7	0.755	7	12.08	7	193.2
10	6	216.0	6	3.326	6	53.09	6	0.850	6	13.59	6	217.3
11	5	237.6	5	3.694	5	58.99	5	0.944	5	15.10	5	241.5
12	4	259.2	4	4.061	4	64.89	4	1.038	4	16.61	4	265.6
13	3	280.8	3	4.317	3	70.78	3	1.132	3	18.12	3	289.8
14	2	302.4	2	4.795	2	76.68	2	1.227	2	19.63	2	313.9
15	1	224.0	1	5.162	1	82.58	1	1.321	1	21.14	1	338.1
16	0	345.6	0	5.530	0	88.47	0	1.416	0	22.65	0	362.2

N = NUMBER OF COUNTS

t = TIME FOR EACH COUNT = 21.6 μsec

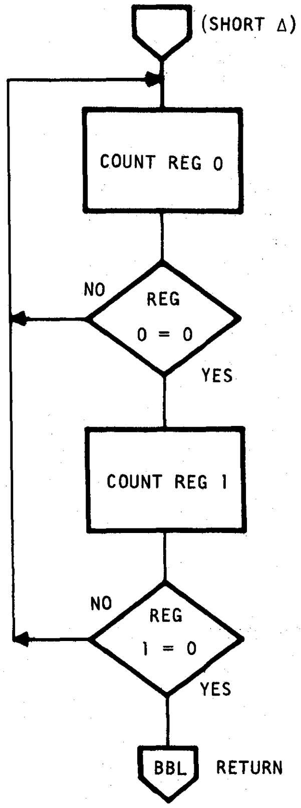
$T_{TOTAL} = T_0 + T_1 + T_2 + T_3 + T_4 + T_5$

Short Delay

The following routine accomplishes a variable time delay with a setable range of 44 microseconds to 5.88 milliseconds, by presetting two registers before the routine is entered. The routine simply uses two cascaded ISZ instructions as previously defined. The procedure for entering this routine as a subroutine requires four instruction words as follows:

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0	—		—		
	1	—		—		
	2	20		FIM	P0	DELAY 44 μSEC TO 5.8 millisec
	3	D0D1		D0	D1	
	4	5X		JMS		
	5	00			(SHORT Δ)	
	6	—		—		
	7	—		—		

Register pair zero is fetched to the data constants from Table 8-3 to give the delay time required.



EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6		3		7
4		2		5
2		1		3
0	COUNT D_0	0	COUNT D_1	1

INDEX REGISTER MAP

FIGURE 8-8

Flow Chart of Short Delay

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	00	70	(SHORT Δ)	ISZ	0	COUNT THE FIRST REGISTER
	01	00		—	(SHORT Δ)	
	02	71		ISZ	1	COUNT THE SECOND REGISTER
	03	00		—	(SHORT Δ)	
	04	C0		BBL	0	RETURN
	05					
	06					
	7					

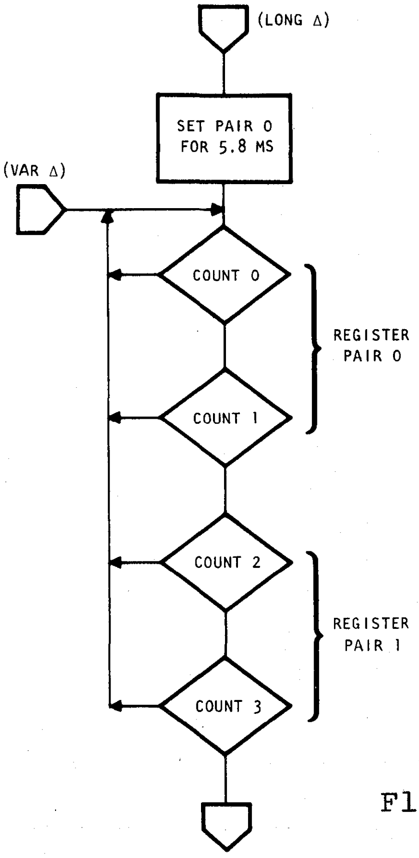
Longer Delays (5.8 milliseconds to 1.5 seconds)

For delays longer than 5.8 milliseconds provided by (Short Δ) the following routine is useful. This routine basically cascades four registers but based on where the routine is entered a couple of variations are possible. By fetching register pair 1 before entering at (Long Δ) and allowing the subroutine to set pair 0 to zero the delay setting has a resolution of 5.8 milliseconds.

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0	—		—		
	1	—		—		
	2	22		FIM	P1	FETCH P1 BEFORE GOING TO (LONG Δ)
	3	$D_2 D_3$		D_2	D_3	
	4	5X		JMS		
	5	10			(LONG Δ)	

By externally setting both pairs 1 and 0 and entering at (VAR Δ) a resolution of 22 microseconds can be obtained.

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0	—		—		
	1	—		—		
	2	20		FIM	P0	FETCH P0 AND P1 BEFORE GOING TO (VAR Δ)
	3	$D_0 D_1$		D_0	D_1	
	4	22		FIM	P1	
	5	$D_2 D_3$		D_2	D_3	
	6	5X		JMS		
	7	12			(VAR Δ)	



EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6		3		7
4		2		5
2	COUNT D_2	1	COUNT D_3	3
0	COUNT D_0	0	COUNT D_1	1

INDEX REGISTER MAP

FIGURE 8-9

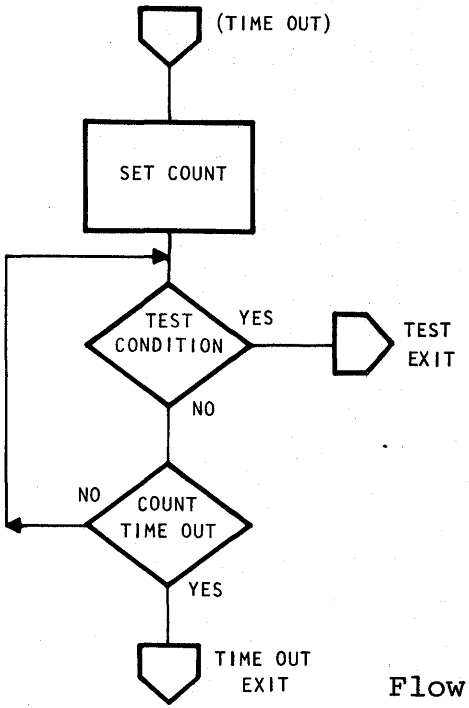
Flow Chart of Longer Delay

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	10	20	(LONG Δ)	FIM	PO	SET REGISTER PAIR O
	11	00		O	O	
	12	70	(VAR Δ)	ISZ	O	COUNT REG. PAIRO (5.8 Millisec)
	13	12			(VAR Δ)	
	14	71		ISZ	1	
	15	12			(VAR Δ)	
	16	72		ISZ	2	COUNT REG. PAIR 1
	17	12			(VAR Δ)	FROM 5.8 Millisec TO 1.5 SEC
	18	73		ISZ	3	
	19	12			(VAR Δ)	
	1A	CO		BBL	O	RETURN

Control Timeout

By interjecting a test condition within a delay loop, a timeout can be affected.

In the example given the simplest test condition is used. If the test condition occurs within the selected time interval the routine will terminate with the TEST EXIT. If the test condition does not occur within the selected time count the routine will terminate with a TIMEOUT EXIT.



EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6		3		7
4		2		5
2		1		3
0	COUNT D_0	0	COUNT D_1	1

INDEX REGISTER MAP

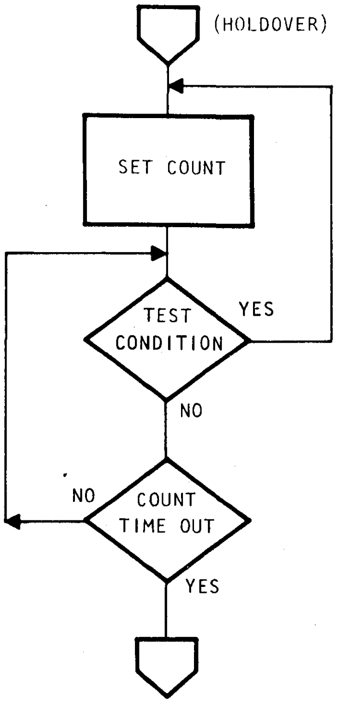
FIGURE 8-10

Flow Chart of Control Timeout

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	20	20	(TIMEOUT)	FIM	PO	SET TIMEOUT COUNT
	21			D_0	D_1	
	22	19	TEST	JCN	T1	EXIT IF CPU TEST INPUT OCCURS
	23				EXIT	
	24	70		ISZ	O	COUNT THE TIMEOUT
	25	22			TEST	
	26	71		ISZ	1	
	27	22			TEST	
	28	CO		BBL	O	RETURN ON TIMEOUT

Holdover

A variation of the timeout subroutine is the holdover, where the timeout count is reset if the test condition occurs.



EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6		3		7
4		2		5
2		1		3
0	COUNT D_0	0	COUNT D_1	1

INDEX REGISTER MAP

FIGURE 8-11
Flow Chart of Holdover

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0		(HOLD OVER)	FIM	PO	SET HOLD OVER COUNT
	1			D_0	D_1	
	2		TEST	JCN	TI	RESET COUNT IF CPU TEST INPUT OCCURS
	3				(HOLD OVER)	
	4			ISZ	0	COUNT TIMEOUT
	5				TEST	
	6			ISZ	1	
	7				TEST	
	8			BEL	O	RETURN WHEN HOLDOVER TIMEOUT OCCURS
	9					
	A					
	B					
	C					

COMPARE SUBROUTINES

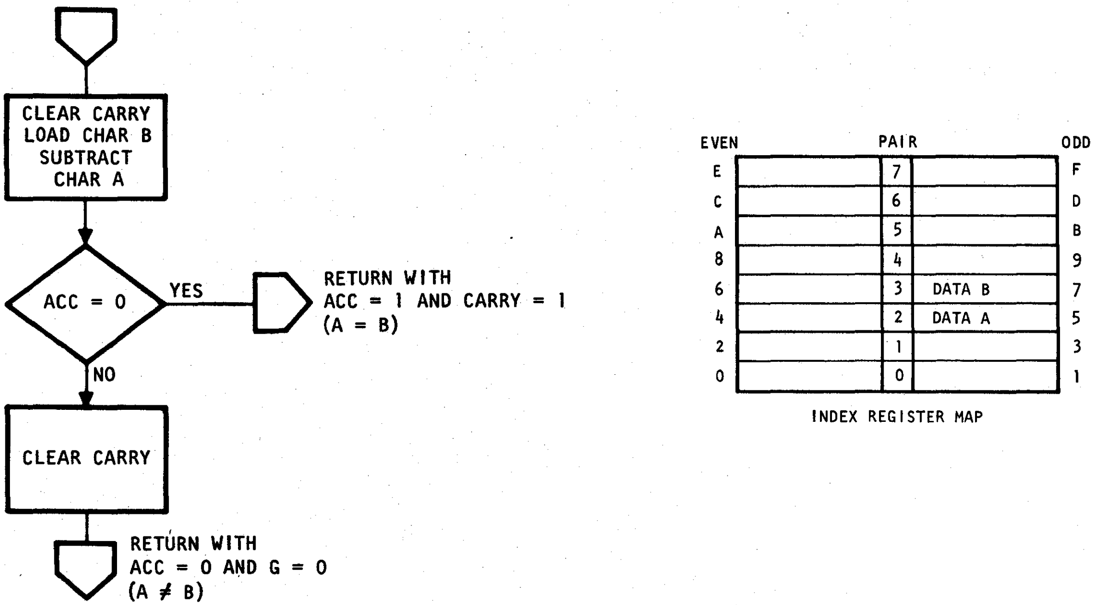
A basic technique in data handling is the use of the SUB instruction to perform data comparisons. If the carry bit is initially cleared and two characters are subtracted the resulting accumulator and carry bit provide the following comparison conditions easily tested using the JCN instruction.

COMPARISON	ACCUMULATOR	CARRY	TEST CONDITION
REG > ACC	≠0	0	CO
REG = ACC	0	1	AO
REG < ACC	≠0	1	A1 · C1
REG ≦ ACC	X	1	C1
REG ≧ ACC	≠0	0	CO or AO
	0	1	
REG ≠ ACC	≠0	X	A1

(Compare) Four Bits

A subroutine which indicates only equal or not-equal conditions is sufficient in many applications. In this example the carry bit is cleared, register 7 is loaded to the accumulator and register 5 is then subtracted from the accumulator. The JCN instruction tests the accumulator and goes to a BBL 1 return if zero, or clears the carry and does a BBLO return if nonzero.

Some features of this routine are that the registers are unaffected and the compare condition is available in either the carry bit or the accumulator.



EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6		3	DATA B	7
4		2	DATA A	5
2		1		3
0		0		1

INDEX REGISTER MAP

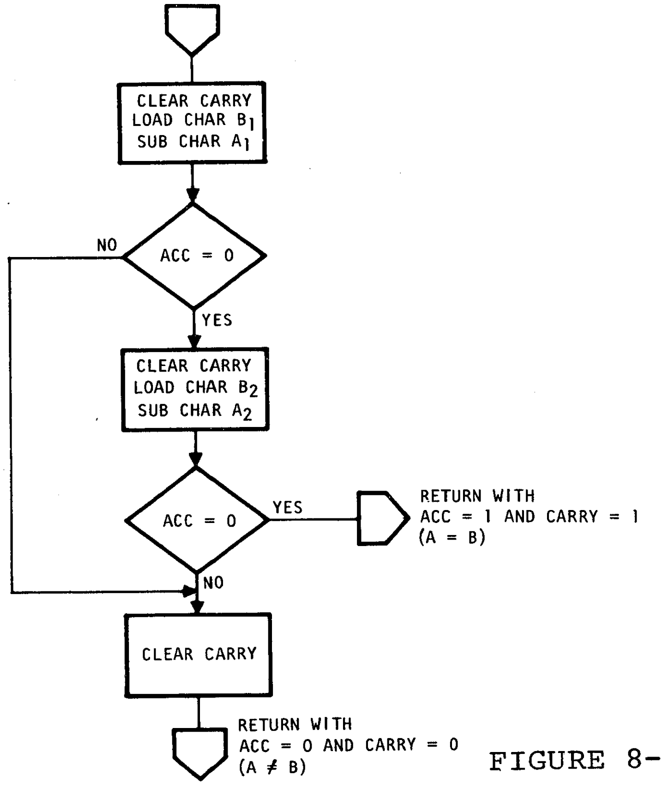
FIGURE 8-12

(Compare) Four Bits

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	80	F1	(COMPARE)	CLC		COMPARE REGISTERS 3 & 5
	81	A3		LD	7	
	82	95		SUB	5	
	83	1C		JCN	A1	
	84	86			NO COMPARE	
	85	C1		BBL	1	C1 OR A1 EQUALS COMPARE
	86	F1	NO COMPARE	CLC		
	87	C0		BBL	0	C0 OR A0 EQUALS NO COMPARE
	8					

(Compare) Eight Bits

A variation of the preceeding routine is presented which provides equal or not-equal comparing of 8 bits. This routine provides the same features as the 4 bit routine and can be extended in increments of 4 bits to any practical length.



EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6	DATA B_2	3	DATA B_1	7
4	DATA A_2	2	DATA A_1	5
2		1		3
0		0		1

INDEX REGISTER MAP

FIGURE 8-13

(Compare) Eight Bits

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	80	F1	(COMPARE)	CLC		COMPARE REGISTERS 3 & 5
	81	A3		LD	7	
	82	95		SUB	5	
	83	1C		JCN	A1	
	84	8B			NO COMPARE	
	85	F1		CLC		COMPARE REGISTERS 4 & 6
	86	A4		LD	6	
	87	96		SUB	4	
	88	1C		JCN	A1	
	89	8B			NO COMPARE	
	8A	C1		BBL	1	C1 OR A1 EQUALS COMPARE
	8B	F1	NO COMPARE	CLC		
	8C	C0		BBL	0	C0 OR A0 EQUALS NO COMPARE

LOGICAL OPERATIONS

Logical operations such as AND, OR, and exclusive OR (XOR) can be performed on either the register bit level or on the program decision level.

For a review of these operations and how they might be used to set, clear, or complement bits within a register see Table 8-4 The CPU instructions set does not include any instructions for directly performing the logical operations AND, OR, and XOR on the index registers. Manipulating bits within the registers can be accomplished using the RAL and RAR instructions to rotate the bits individually into the carry where they can be set, cleared, or complemented with the STC CLC or CMC instructions.

TABLE 8-4

Boolean Laws of Operation for 0 and 1

OR (+)

$0 + X = X$ meaning: if $X = 1$, $0 + 1 = 1$, if $X = 0$, $0 + 0 = 0$
$1 + X = 1$ meaning: if $X = 1$, $1 + 1 = 1$; if $X = 0$, $1 + 0 = 1$

AND (•)

$0 \cdot X = 0$ meaning: if $X = 1$, $0 \cdot 1 = 0$; if $X = 0$, $0 \cdot 0 = 0$
$1 \cdot X = X$ meaning: if $X = 1$, $1 \cdot 1 = 1$; if $X = 0$, $1 \cdot 0 = 0$

XOR (⊕)

$0 \oplus X = X$ meaning: if $X = 1$, $0 \oplus 1 = 1$; if $X = 0$, $0 \oplus 0 = 0$
$1 \oplus X = \overline{X}$ meaning: if $X = 1$, $1 \oplus 1 = 0$; if $X = 0$, $1 \oplus 0 = 1$

Example of four bit operations

Operand	XXXX	XXXX	XXXX
Operator	OR 1101	AND 1101	XOR 1101
Result	11X1	XX0X	$\overline{XX}X\overline{X}$

Summary

OR = Set any bits where the OR operator equals 1
AND = Reset any bits where the AND operator equals 0
XOR = Complement any bits where the XOR operator equals 1

An example might be where a 4 bit register is used to store, up to 4 individual flags for remembering the occurrence of random functions. An example is given where such a flag register is loaded to the accumulator and rotated to the desired bit which is cleared (set or complemented) and then rotated back into position and restored in the flag register.

When using register bits 8 or 4 for flags, use a RAL/RAR sequence to save program steps. Likewise, when using bits 2 or 1 for flags, use a RAR/RAL sequence.

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0			—		
	1			—		
	2	AA		LD	A	LOAD FLAG REGISTER TO ACCUMULATOR
	3	F5		RAL		⊤ ROTATE BIT 4 TO CARRY
	4	F5		RAL		↓
	5	FA		STC		SET THE FLAG
	6	F6		RAR		⊤ ROTATE BIT 4 BACK
	7	F6		RAR		↓
	8	BA		XCH	A	RESTORE FLAG REGISTER
	9			—		
	A			—		
	B					

Performing logical operations on the programming level is accomplished using combinations of the two decision instructions JCN and ISZ. These instructions are combined in sequences to perform programmed decisions such as AND, OR, and XOR; and their dual operations NAND, NOR and compare.

The AND and its dual, NOR, are accomplished as shown in Figure 8-14 where two positive decisions are required to give A · B. The OR and its dual, NAND, function are accomplished as shown in Figure 8-14 where either of two decisions being positive gives A + B.

The XOR and its dual, compare, function require three decisions as shown in Figure 8-14 where if A is a positive decision, B must be a negative decision, or if A is a negative decision, B must be a positive decision to give the result $\bar{A} \cdot B + A \cdot \bar{B}$.

An example of a programmed AND is given where N passes AND flag 8 are required to continue in sequence.

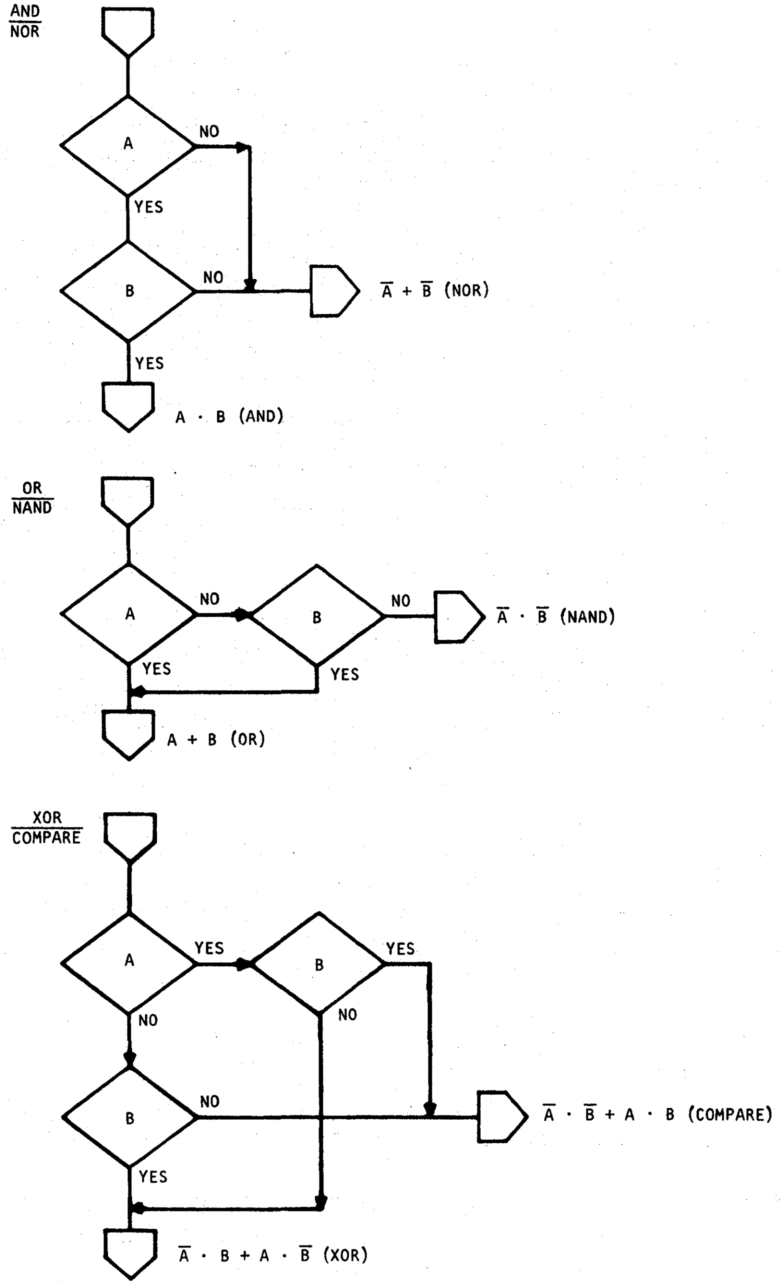

AND / NOR
A
NO
YES
B
NO
$\overline{A} + \overline{B}$ (NOR)
YES
$A \cdot B$ (AND)
OR / NAND
A
NO
B
NO
$\overline{A} \cdot \overline{B}$ (NAND)
YES
YES
$A + B$ (OR)
XOR / COMPARE
A
YES
B
YES
NO
NO
B
NO
$\overline{A} \cdot \overline{B} + A \cdot B$ (COMPARE)
YES
$\overline{A} \cdot B + A \cdot \overline{B}$ (XOR)

FIGURE 8-14

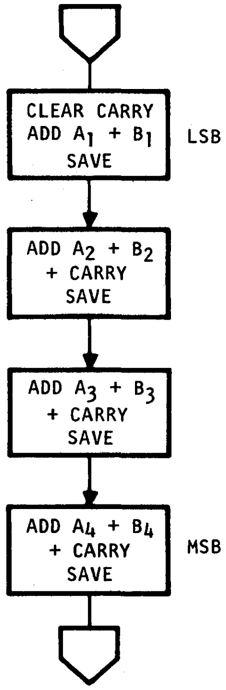
Logical Operations

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0			—		
	1			—		
	2			—		
	3			ISZ	7	N PASSES
	4				NOT N	
	5			LD	A	FLAG 8
	6			RAL		
	7			JCN	CO	
	8				NOT FLAG8	
	9			—		EXECUTE IF N PASSES AND FLAG 8
	A			—		
	B			—		
	C			—		

ADDITION

Addition of two 4 bit numbers can be done directly by the ADD or ADM instructions. The ADD instruction adds from one of the index registers to the accumulator and the ADM adds from the selected RAM register into the accumulator.

Numbers longer than 4 bits can be added in multiples of 4 bits. This technique is referred to as multiple precision arithmetic. The carry bit automatically maintains the carry/link between each group of 4 bits to be added. The example shows a routine for adding two 16 bit binary numbers. Note that the first step clears the carry bit. The least significant bits are added first so that the carry/link will propagate. If an overflow occurs the carry bit will contain a 1 at the end of the routine.



EVEN		PAIR		ODD
E		7		F
C		6		D
A	A_4	5	A_3	B
8	A_2	4	A_1	9
6		3		7
4	B_4	2	B_3	5
2	B_2	1	B_1	3
0		0		1

INDEX REGISTER MAP

FIGURE 8-15

Multiple Precision Addition

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0	F1	(ADD)	CLC		ADD AB89 TO 4523
	1	A9		LD	9	ADD REG 9 TO REG 3
	2	83		ADD	3	
	3	B3		XCH	3	
	4	A8		LD	8	ADD REG 8 TO REG2 WITH CARRY
	5	82		ADD	2	
	6	B2		XCH	2	
	7	AB		LD	B	ADD REG B TO REG5 WITH CARRY
	8	85		ADD	5	
	9	B5		XCH	5	
	A	AA		LD	A	ADD REG A TO REG4 WITH CARRY
	B	84		ADD	4	
	C	B4		XCH	4	
	D	C0		BBL	0	RETURN

MULTIPLICATION

The two methods of multiplication are the brute-force method and long-hand method.

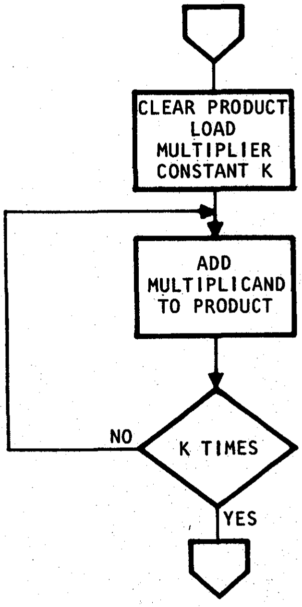
Brute-Force Method

Multiply is accomplished with repeated addition. Beginning with a number to be multiplied (Multiplicand) and a number to multiply by (Multiplier) the brute-force method adds the multiplicand repeatedly into the product, doing the addition as many times as designated by the multiplier. This method is sufficient and sometimes appropriate for small numbers but can take considerable time for large numbers.

One example of the brute-force method of multiply is given where the multiplicand is multiplied by a constant K. The routine clears the product registers, sets the multiplier to K and then adds the multiplicand to the product, K times. Since the ISZ instruction is used to count K, the complement plus one must be used for the constant. Table 8-2 is useful for determining these constant. It should be noted that this routine performs the execute/count sequence as defined in the section on counting.

16 BITS	A	B	8	9	Multiplicand
8 BITS	X		6	7	Multiplier
16 BITS	4	5	2	3	Product

Registers used in multiply examples



EVEN		PAIR		ODD
E		7		F
C		6		D
A	MPLICAND	5	MPLICAND	B
8	MPLICAND	4	MPLICAND	9
6	MULT K	3	MULT K	7
4	PROD	2	PROD	5
2	PROD	1	PROD	3
0		0		1

INDEX REGISTER MAP

FIGURE 8-16

Brute-Force Method of Multiplication

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	A 0	22	[MULT by K]	FIM	P1	CLEAR THE PRODUCT
	A 1	00		0	0	
	A 2	24		FIM	P2	
	A 3	00		0	0	
	A 4	26		FIM	P3	FETCH MULTIPLIER, K
	A 5	D_6D_7		D_6	D_7	
	A 6	5X	ADDKTIMES	JMS		ADD MULTIPLICAND TO PRODUCT
	A 7				(ADD)	
	A 8	77		ISZ	7	DO K ADDITIONS
	A 9	A6			ADD K TIMES	
	A A	76		ISZ	6	
	A B	A6			ADD K TIMES	
	A C	C0		BBL	0	RETURN

Long-Hand Method

Consider an example of multiplying six by seven.

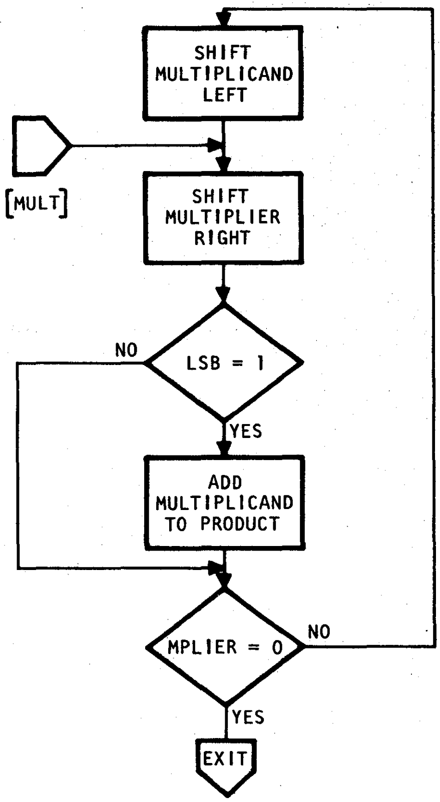
The brute-force method suggests adding the number 6 to the product, 7 times for the result. Now consider the example again as shown done in the long-hand method.

Multiplicand	0110	=	6		
Multiplier	x) 0111	=	7		
	0110	=	1 x 6	=	6
	01100	=	2 x 6	=	12
	011000	=	4 x 6	=	24
	+) 0000000	=	8 x 0	=	0
Product	101010	=			42

When done by the long-hand method only three additions are needed. Each bit position of the multiplier containing a 1, adds the multiplicand times the multiplier bit position weight. Multiplying the multiplicand by the bit position weight is accomplished by a left shift operation.

An example of the long-hand method of multiply is given where the multiplier can be variable. The routine shifts the multiplier right and tests the LSB in the carry bit. If the LSB is 1, the multiplicand is added to the product. If the LSB is 0 the addition is skipped. The multiplier is tested for all zeroes to determine completion. If the operation is not complete the multiplicand is shifted left one place to multiply it by the current bit position weight. The routine then proceeds as above testing the LSB and adding until the multiplier becomes all zeros.

This routine assumes the product area is initially cleared. In addition, overflow is stored in the carry bit and can be tested by the main program. The multiplier and multiplicand are not saved. Note that the entry point [MULT] is not at the beginning. The (ADD) subroutine is given in the section on addition.



EVEN		PAIR		ODD
E		7		F
C		6		D
A	MPLICAND	5	MPLICAND	B
8	MPLICAND	4	MPLICAND	9
6	MPLIER	3	MPLIER	7
4	PROD	2	PROD	5
2	PROD	1	PROD	3
0		0		1

INDEX REGISTER MAP

FIGURE 8-17
Long-Hand Method
of Multiplication

ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
C0	5X	NEXT BIT	JMS		SHIFT MULTIPLICAND LEFT
C1	E0			(AB89 ←)	
C2	5X	[MULT]	JMS		SHIFT MULTIPLIER RIGHT
C3	F0			(67 →)	
C4	/A		JCN	C0	IS MULTIPLIER BIT = 0
C5	C8			ZERO	
C6	5X		JMS		ADD MULTIPLICAND TO PRODUCT
C7				(ADD)	IF MULTIPLIER BIT = 1
C8	A7	ZERO	LD	7	IS REMAINING MULTIPLIER = 0
C9	/C		JCN	A1	
CA	C0			NEXT BIT	
CB	A6		LD	6	
CC	/C		JCN	A1	
CD	C0		.	NEXT BIT	
CE	C0		BBL	0	EXIT WHEN MULTIPLIER IS ALL ZEROES

ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
E0	F1	(AB89 ←)	CLC		SHIFT REGISTERS AB89 LEFT
E1	A9		LD	9	ONE POSITION
E2	F5		RAL		
E3	B9		XCH	9	
E4	A8		LD	8	
E5	F5		RAL		
E6	B8		XCH	8	
E7	AB		LD	B	
E8	F5		RAL		
E9	BB		XCH	B	
EA	AA		LD	A	
EB	F5		RAL		
EC	BA		XCH	A	
ED	C0		BBL	0	
EE					
EF					
F0	F1	(67 →)	CLC		SHIFT REGISTERS 6 & 7 RIGHT
F1	A6		LD	6	ONE POSITION
F2	F6		RAR		
F3	B6		XCH	6	
F4	A7		LD	7	
F5	F6		RAR		
F6	B7		XCH	7	
F7	C0		BBL	0	

SQUARE ROOT

There are various formulas for approximating the square root of a number. There is also the long-hand division technique learned in grade school. As shown in the example using a decimal number the technique is to first separate the number into pairs of digits. A trial divisor is then selected for the most significant pair. When one is found that gives zero or the smallest positive remainder it is saved as a partial result. This partial result is doubled and multiplied by ten to become the basis for a new trail divisor. A new trial divisor digit is added to the doubled partial result. The new trail divisor operates on a new partial remainder, again looking for zero or the smallest positive remainder. The new partial remainder consists of the division remainder plus the next two digits of the dividend.

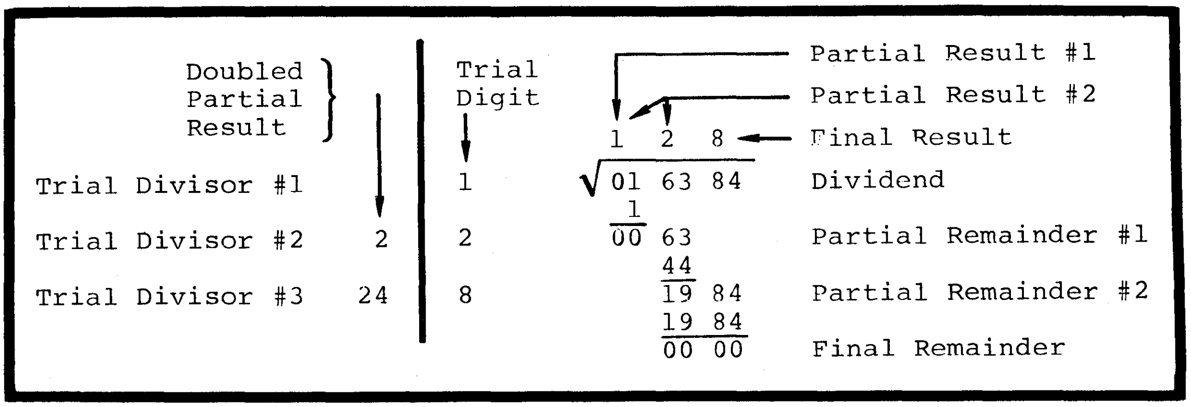


FIGURE 8-18

Example of Decimal Long-Hand Square Root

The long-hand square root technique also works for binary numbers and in fact is simpler because of the binary operations. In binary there are only two trial choices, 1 or 0 and to double the partial result is simply a shift left, as is multiplying the partial result by the number base. Also, the final result can be derived from the trial division by shifting right one place at the end of the operation. This allows the quotient and divisor to use the same register.

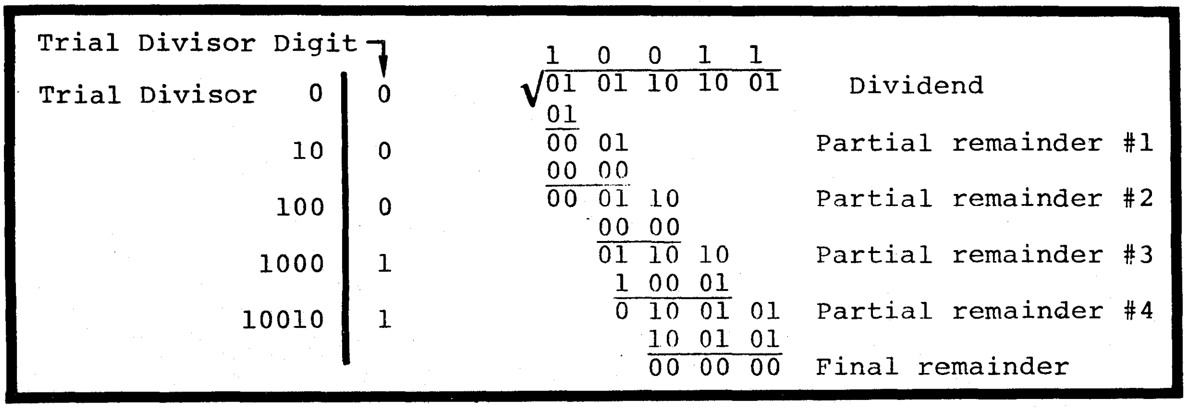


FIGURE 8-19

Example of Binary Long-Hand Square Root

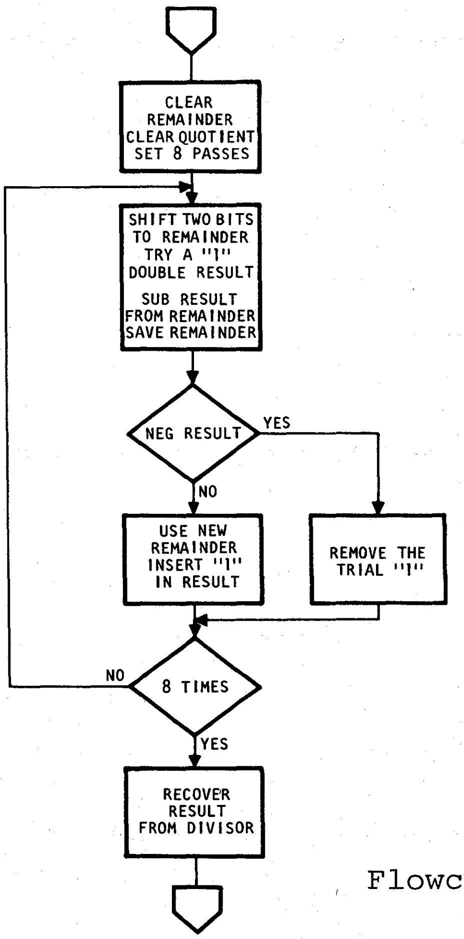
A programming example for the square root of a 16 bit integer is given. The routine initializes the remainder and quotient and sets a pass counter for the 8 pairs of the 16 bit integer. Two bits of the integer are shifted into the partial remainder area where the trial divisor is subtracted. Before the subtractions, the trial divisor is doubled and set to "1". If the subtraction gave a positive result the new partial remainder is saved and the trial "1" is inserted into the combined quotient-divisor as a result. If the subtraction gave a negative result, the trial "1" bit is removed from the divisor.

The process of shifting two bits into the remainder and subtracting the trial divisor is repeated for 8 passes. When the operation is complete, the trial divisor is shifted right one place to obtain the quotient.

This subroutine requires approximately 8.3 milliseconds to execute when the CPU clock is 10.8 microseconds.

DIVISOR & QUOTIENT		
7	4	5

REMAINDER			DIVIDEND			
F	C	D	A	B	8	9



EVEN		PAIR		ODD
E		7	REMAINDER	F
C	REMAINDER	6	REMAINDER	D
A	DIVIDEND	5	DIVIDEND	B
8	DIVIDEND	4	DIVIDEND	9
6		3	DIVISOR	7
4	DIVISOR	2	DIVISOR	5
2	NEW REMAINDER	1	NEW REMAINDER	3
0		0	COUNT	1

INDEX REGISTER MAP

FIGURE 8-20

Flowchart for Long-Hand
Square Root

Program for square root subroutine

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0		[SQ ROOT]	FIM	P7	CLEAR REMAINDER
	1			0	0	
	2			FIM	P6	
	3			0	0	
	4			FIM	P3	CLEAR QUOTIENT
	5			0	0	
	6			FIM	P2	
	7			0	0	
	8			FIM	P0	SET FOR 8 PASSES
	9			0	8	
	A		TWO MORE	JMS		SHIFT TWO BITS INTO SUB AREA (REMAINDER)
	B				(FCDAB89←)	
	C			JMS		
	D				(FCDAB89←)	
	E			STC		TRY A "1" TRIAL DIGIT
	F			JMS		MULTIPIY PARTIAL RESULT BY 2
	0				(745←)	
	1			CLC		SUBTRACT PARTIAL RESULT FROM REMAINDER
	2			LD	D	
	3			SUB	5	
	4			XCH	3	SAVE NEW REMAINDER
	5			CMC		
	6			LD	C	
	7			SUB	4	
	8			XCH	2	SAVE NEW REMAINDER
	9			CMC		
	A			LD	F	
	B			SUB	7	
	C			JCN	C0	CHECK FOR NEGATIVE RESULT
	D				NEG	
	E			XCH	F	USE NEW REMAINDER IF POS RESULT
	F			LD	2	
	0			XCH	C	
	1			LD	3	
	2			XCH	D	
	3			INC	5	INSERT A "1" IN RESULT
	4		COUNT	ISZ	1	DO 8 PASSES
	5				TWO MORE	
	6			JMS		SHIFT TRIAL DIVISOR RIGHT FOR FINAL
	7				(745→)	RESULT
	8			BBL	0	EXIT
	9		NEG	LD	5	REMOVE THE TRIAL "1" IF NEG RESULT
	A			DAC		
	B			XCH	5	
	C			JUN		
	D				COUNT	

Subroutines used in [square root]

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0		(FCDAB89←)	CLC		SHIFT REGISTERS FCDAB89
	1			LD	9	LEFT ONE POSITION
	2			RAL		
	3			XCH	9	
	4			LD	8	
	5			RAL		
	6			XCH	8	
	7			LD	B	
	8			RAL		
	9			XCH	B	
	A			LD	A	
	B			RAL		
	C			XCH	A	
	D			LD	D	
	E			RAL		
	F			XCH	D	
	0			LD	C	
	1			RAL		
	2			XCH	C	
	3			LD	F	
	4			RAL		
	5			XCH	F	
	6			BBL	0	
	7					

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0		(745←)	CLC		SHIFT REGISTERS 745
	1			LD	5	LEFT ONE POSITION
	2			RAL		
	3			XCH	5	
	4			LD	4	
	5			RAL		
	6			XCH	4	
	7			LD	7	
	8			RAL		
	9			XCH	7	
	A			BBL	0	
	B					
	C					
	D					
	E					
	F					
	0		(745 →)	CLC		SHIFT REGISTER 745
	1			LD	7	RIGHT ONE POSITION
	2			RAR		
	3			XCH	7	
	4			LD	4	
	5			RAR		
	6			XCH	4	
	7			LD	5	
	8			RAR		
	9			XCH	5	
	A			BBL	0	
	B					

TELETYPE

The Pro-Log teletype interface shown in Figure 8-21 is a three circuit, six-wire connection which allows full-duplex send and receive, and remote reader control. The three, two-wire circuits are:

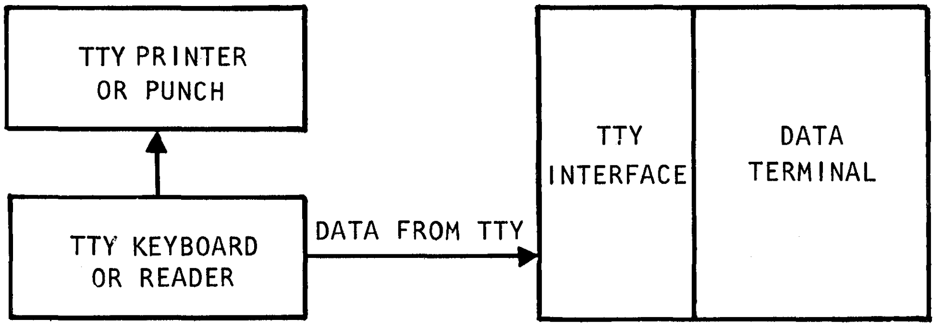
(1) Data to TTY
(2) Data from TTY
(3) Reader Control to TTY

This interface can be used in the following modes with Appropriate TTY Configurations:

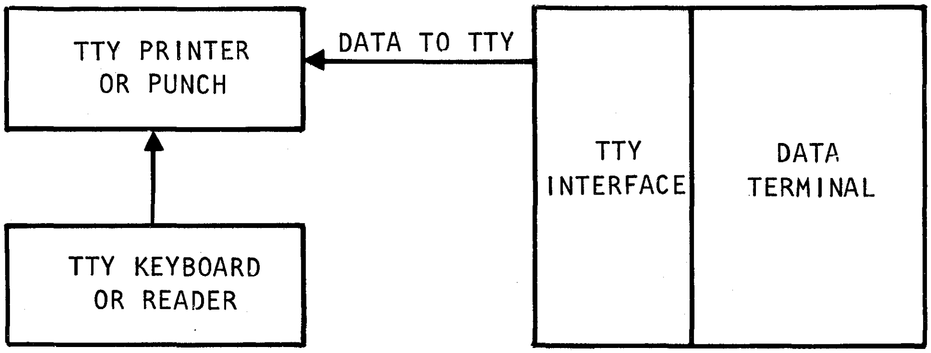
1. Simplex receive
2. Simplex send
3. Full duplex
4. With or without Remote Reader Control

TTY MODES

Simplex Send mode allows the TTY to be used as an input device in LINE operation.



Simplex Receive mode allows the TTY to be used as an output device in LINE operation.



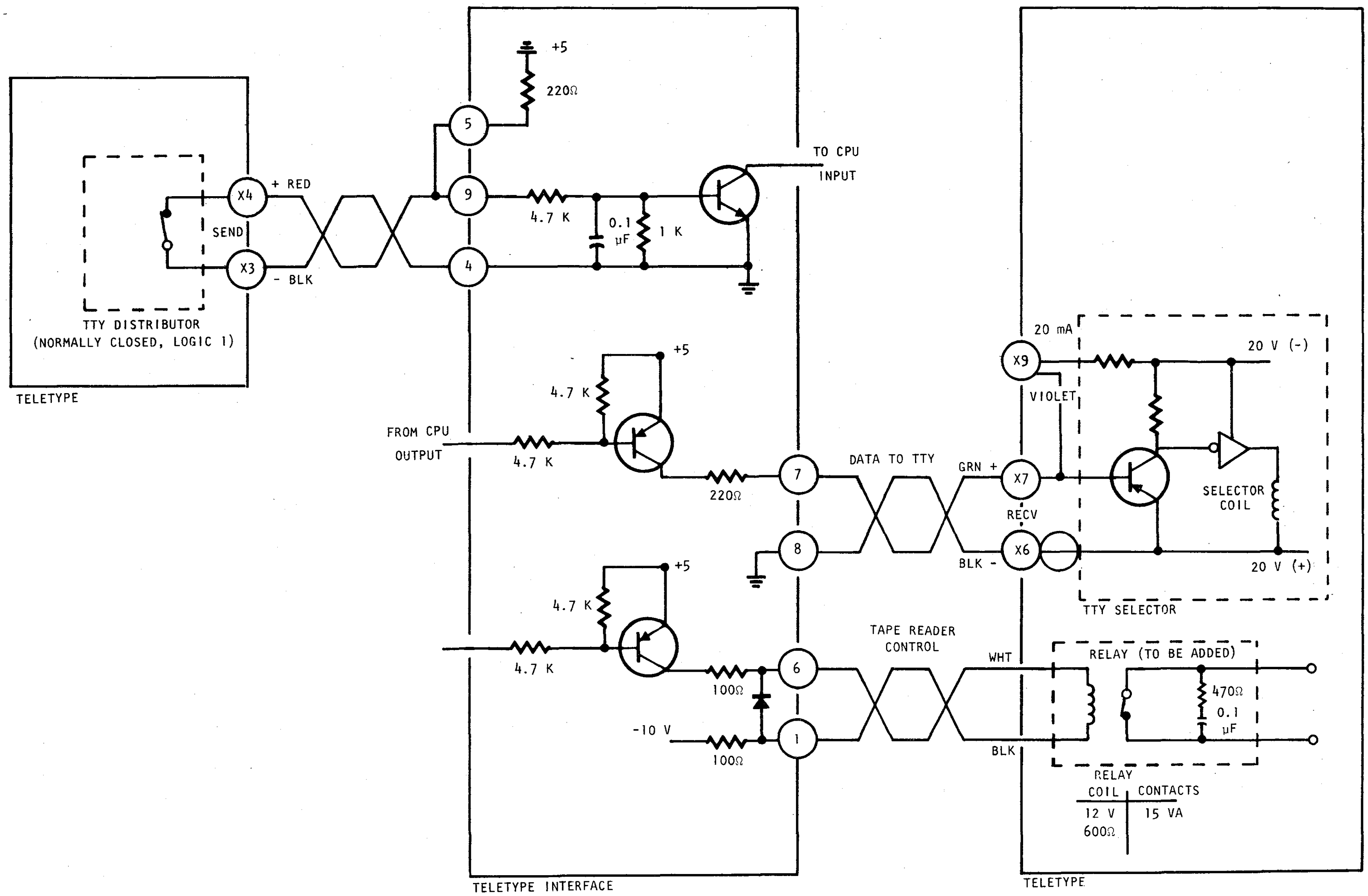
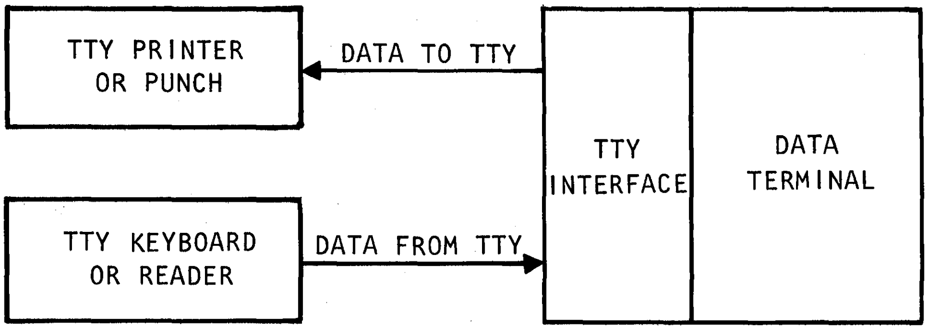


FIGURE 8-21
Teletype Interface

Full-Duplex mode allows the TTY to be used as an input and output device in LINE operation. The separate send and receive allows the data terminal to edit the input data before printing and/or punching the output data.



Remote Reader mode allows the TTY paper tape reader to be activated by a remote device in LINE operation. This mode would be used in conjunction with Simplex Send or Full-Duplex operation. In Full-Duplex, the remote reader control allows paper tape editing.

TTY REQUIREMENTS

The full PRO-LOG TTY interface requires specific teletype configurations. The general configuration requires a TTY modified for Full-Duplex and Remote Reader Control. Specifically there are three circuit connection requirements for the TTY:

(1) 20 mA neutral loop send
(2) 20 mA neutral loop receive
(3) 15 volt neutral loop reader control

TTY FORMAT

The programming examples given here assume a serial by character, serial by bit TTY data format as shown in Figure 8-22. The character structure consists of a minimum of ten equal time intervals; one start bit, 8 data bits and at least one stop bit. When the TTY is in a stopped state the line is held in the logical 1 state. The first transition on the line is always a start bit (logical 0).

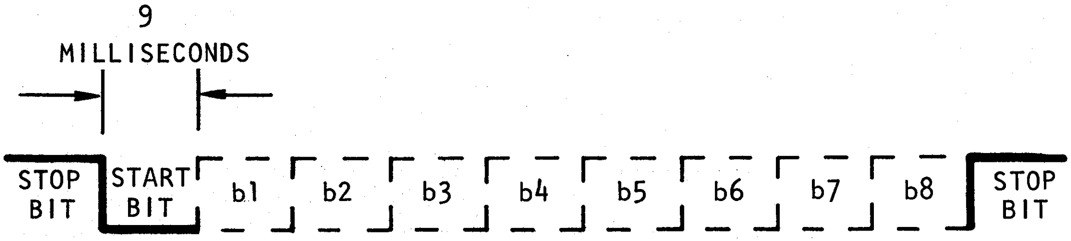


FIGURE 8-22

TTY Data Line Format

The character set assumed is 7 level ASC II plus parity with the codes defined as in Figure 8-23. The bit sequence is least significant bit first, to most significant in ascending consecutive order. A character parity bit follows the most significant bit of the data character. The character parity is assumed to be even over the eight bits i.e., an even number of 1 bits per character including the parity bit.

With a full duplex TTY interface various methods can be used when reading. The most common method is to simply echo each bit back to the TTY as it is detected and print the incoming character immediately. Another method is to completely read the entire TTY character and then issue a separate print response to the TTY only if desired. This allows the incoming data to be edited before printing.

TTY Read Without Echo

In this example the program is written to receive from the TTY without echoing a print response. The program reads the TTY input line searching for a start bit. When the start bit occurs, the program clocks off 4.5 milliseconds of delay and then makes eight periodic samples, one every 9 milliseconds. At each 9 milliseconds sample the TTY input is read as either a 1 or 0 bit. The eight serial bits are assembled into the ASC II character by shifting the bits inot a data storage area. When the eight bits have been read, an additional 9 milliseconds interval is generated to prevent the program from returning to a subsequent read too early.

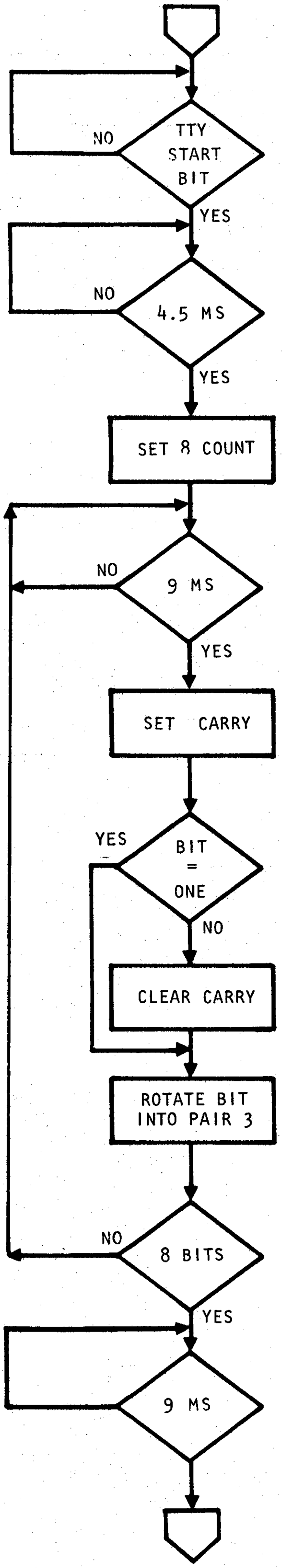
Bits	b7 b6 b5 →				0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4 t	b3 t	b2 t	b1 t	Column › Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	| ¦
1	1	0	1	13	CR	GS	–	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

Control Characters

NUL	Null
SOH	Start of Heading
STX	Start of Text
ETX	End of Text
EOT	End of Transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell (audible or attention signal
BS	Backspace
HT	Horizontal Tabulation (punched card skip)
LF	Line Feed
VT	Vertical Tabulation
FF	Form Feed
CR	Carriage Return
SO	Shift Out
SI	Shift In
DLE	Data Link Escape
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4 (Stop)
NAK	Negative Acknowledge
SYN	Synchronous Idle
ETB	End of Transmission Block
CAN	Cancel
EM	End of Medium
SUB	Substitute
ESC	Escape
FS	File Separator
GS	Group Separator
RS	Record Separator
US	Unit Separator
DEL	Delete

FIGURE 8-23

TTY Character Set



EVEN		PAIR		ODD
E		7		F
C		6		D
A		5		B
8		4		9
6	TTY MSD	3	TTY LSD	7
4		2		5
2	DELAY	1	DELAY	3
0	DELAY	0	DELAY	1

INDEX REGISTER MAP

FIGURE 8-24

Flow Chart for TTY Read without Echo

Mnemonic Listing for TTY Read

	ADR	INSTR	LABEL	MNEMONIC	OPERAND	COMMENTS
	0		[RD TTY]	JCN	TO	FIND START BIT
	1				[RD TTY]	
	2			FIM	PO	DELAY ONE-HALF BIT TIME
	3			7	3	
	4			JMS		
	5				(SHORT Δ)	
	6			FIM	P2	SET COUNT FOR 8 BITS
	7			0	8	
	8		NEXT BIT	JMS		DELAY ONE BIT TIME
	9				(9 MS)	
	A			CLB		CLEAR BIT
	B			JCN	TI	READ ONE TTY BIT
	C				NO BIT	
	D			STC		SET BIT
	E		NO BIT	JMS		ROTATE DATA BITS INTO PAIR 3
	F				(ROT P3→)	
	0			ISZ	5	READ 8 BITS
	1				NEXT BIT	
	2			JMS		STOP BIT DELAY
	3				(9 MS)	
	4			BBL	0	
	5					
	6					
	7					
	8					
	9		(ROT P3→)	LD	6	SHIFT REG 6 & 7 RIGHT ONE PLACE
	A			RAR		
	B			XCH	6	
	C			LD	7	
	D			RAR		
	E			XCH	7	
	F			BBL	0	

APPENDIX A

SYMBOLS AND DEFINITIONS

The symbols and definitions presented in this appendix are used throughout PRO-LOG documentation.

BACKGROUND INFORMATION

Physical and Active States in Binary Logic (0 and 1)

Binary logic implies a two-state system. In physical applications of binary logic, the two states can be any meaningful pair of physical states such as high/low, positive/negative, in/out, up/down, etc that will either cause a function to happen or to not happen.

Binary Operations

Binary logic application is governed by the rules of Boolean Algebra. There exist only three operations in Boolean Algebra; AND, OR, NOT (invert).

AND

AND implies a combination of two or more active conditions to achieve a result.

OR

OR implies a choice between two or more active conditions to achieve a result.

NOT

NOT implies the negative or inverse. Since there are only two states in binary logic, the inverse of a function must always assume the opposite state, thus $\overline{0} = 1$, $\overline{1} = 0$.

Duality of Operations

The inherent property of the NOT operation establishes a dual relationship between the AND and OR operations. The dual relationship is such that the AND and OR functions can be interchanged if the active conditions (0 and 1 states) are inverted. This property is stated as De Morgan's theorem in Boolean Algebra.

The importance of this property in physical systems is that the active state can and will assume either physical state if the logic operations have been assembled to achieve a result.

DEFINITIONS

Logic Block Diagram

A logic diagram is one which depicts logic functions with no reference to physical implementations. It consists primarily of logic symbols and is used to depict all logic relationships as simply and understandably as possible. Nonlogic functions are not normally shown. This basic logic diagram is used for educational purposes. The purpose of the logic block diagram is to communicate the overall system concept.

Detailed Logic Diagram

A diagram that depicts all logic functions and also shows nonlogic functions, locations, pin numbers, test points, and other physical elements necessary to describe the physical and functional aspects of the logic is a detailed logic diagram. The detailed logic diagram is used primarily to facilitate the rapid diagnosis and localization of equipment malfunctions. It also is used to verify the physical consistency of the logic and to prepare fabrication instructions. The symbols can be connected by lines that represent signal paths or can be cross-referenced through the use of mnemonic identifiers.

Logic Function

A logic function is a combinational, storage, delay, or sequential function expressing a relationship between signal input(s) to a system or device and the resultant output(s). Logic functions are expressed graphically with the use of logic symbols.

Logic Symbol

A logic symbol is the graphic representation of a logic function.

Symbol Orientation

The orientation of a symbol on a diagram does not alter the meaning of the symbol. However, logic diagrams indicate direction of signal flow by symbol orientation and should, therefore, be logically oriented, consistent with the overall information flow.

Symbol Line Thickness

The weight of a line does not affect the meaning of a symbol. In specific cases, a heavier line may be used for emphasis.

Symbol Size

A symbol may be drawn to any proportional size that suits a drawing, depending on the reduction or enlargement anticipated. Relative sizes of the symbols should be equivalent for related functions.

Table of Combinations

For purposes of this standard, tables of combinations describe the active input/output conditions of the basic logic functions; i.e., HIGH (H) more positive, and LOW (L) relatively less positive or negative.

Identifiers

Identification information is required on and adjacent to logic symbols to specify unique location of logic function on the drawing, within the equipment and its circuit diagram. Identification is required for clarity as follows:

a. Notations shall be placed about the periphery of symbols to identify input and output pin numbers and test points.

b. Line conditions, signal routing, etc may also be labeled for clarity.

c. Details such as stylized waveforms and timing durations may be included when required for clarity.

Mnemonic Identifiers

A mnemonic identifier is a name given to a logic function output for the purpose of cross-reference identification. It is usual practice to assign a meaningful name for the purpose of implying what function is being accomplished. These identifiers can be words, abbreviations, word-number combinations, numbers or symbols. In all cases when mnemonic identifiers are used, they must always appear identically written.

Signal Flow Direction

Logic Diagrams indicate direction of signal flow by symbol orientation, preferred signal flow direction is from left to right. For increased clarity, arrows superimposed on lines may be used. However, arrowheads shall not be placed immediately adjacent to any graphic symbol input or output.

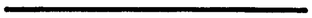
Stylized Waveforms

Stylized waveforms may be placed adjacent to signal lines (where required) to indicate the nature and timing of the signals.

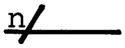
SYMBOLS

Line Symbols

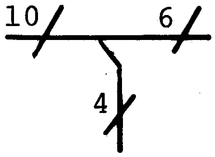
Single Channel Path



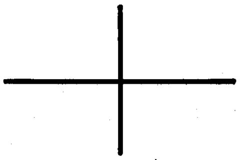
Multiple Channel Paths

 n = Number of Channels

Example: Multiple Channel Paths With Junction

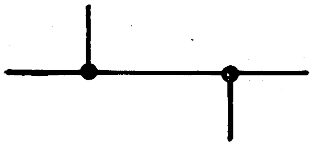


Signal Paths Crossing With no Connection (not necessarily perpendicular)

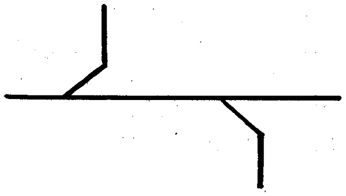


Junction of Signal Paths

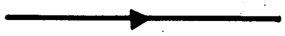
Single Paths:



Multiple Paths:



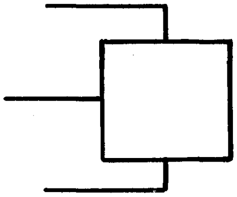
Signal Flow



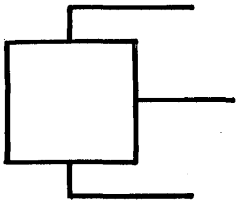
Inputs and Outputs

Preferred (left to right signal flow, no arrows required)

Inputs

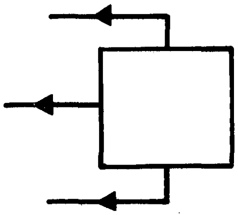


Outputs

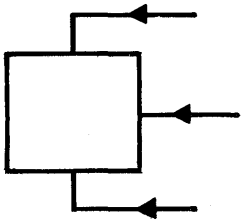


Undesirable But Acceptable (right to left signal flow, arrows required for clarity)

Outputs



Inputs



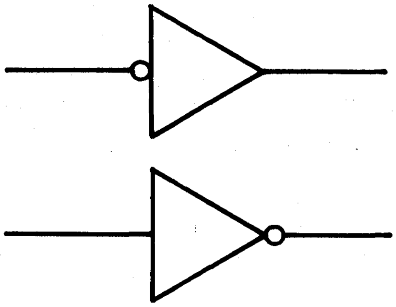
Logic Symbols for Binary Operations

Low Level State Indicators

A small circle symbol at any input or output of a function is used to represent the active low state. A small circle at the input indicates that the relatively low (L) input signal activates the input. Conversely, the absence of a small circle indicates that the relatively high (H) input signal activates the input. A small circle at a symbol output indicates that when the function is activated the output terminal is relatively low (L).

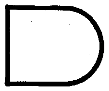
NOT (Invert)

The NOT function is implied when a high input activates a low output or a low input activates a high output. This is represented in its simplest form using the appropriate symbol below. The invert function using low level state indicators applies to AND, OR, and other more complex symbols.



AND

The symbol shown below represents the AND function. The AND symbol can be used with active low state indicators as shown in Table A-1. The AND output is active only if all inputs are active.



OR

The symbol shown below represents the OR function. The OR symbol can be used with active low state indicators as shown in Table A-1. The OR output is active only is any one or more inputs is active.

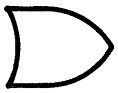
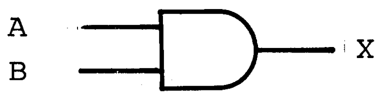
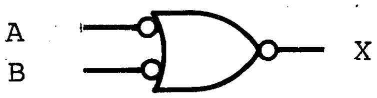
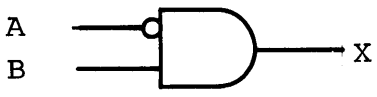
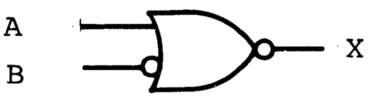
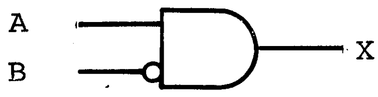
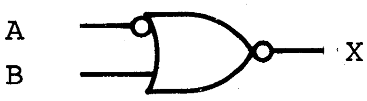
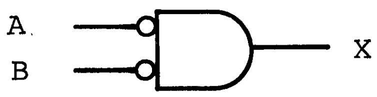
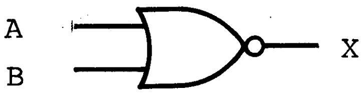
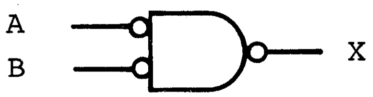
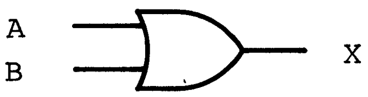
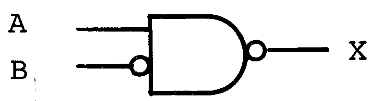
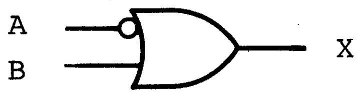
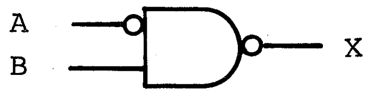
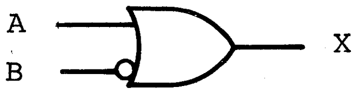
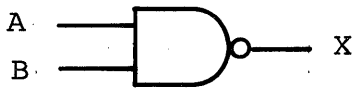
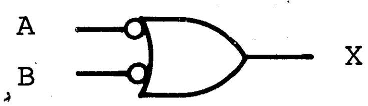


TABLE A-1

Table of Combinations

The following table of combinations illustrates the applications and functions of two variables illustrating duality and use of low level state indicators.

<table>
<tr><th>AND</th><th>OR</th><th>A</th><th>B</th><th>X</th></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>H
L
L
L</td></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>L
L
H
L</td></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>L
H
L
L</td></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>L
L
L
H</td></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>H
H
H
L</td></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>H
L
H
H</td></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>H
H
L
H</td></tr>
<tr><td> </td><td> </td><td>H
H
L
L</td><td>H
L
H
L</td><td>L
H
H
H</td></tr>
</table>

APPENDIX B

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 45
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX C

HEXADECIMAL/DECIMAL INTEGERS

Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	4,294,967,296	1	268,435,456	1	16,777,216	1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	8,589,934,592	2	536,870,912	2	33,554,432	2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	12,884,901,888	3	805,306,368	3	50,331,648	3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	17,179,869,184	4	1,073,741,824	4	67,108,864	4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	21,474,836,480	5	1,342,177,280	5	83,886,080	5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	25,769,803,776	6	1,610,612,736	6	100,663,296	6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	30,064,771,072	7	1,879,048,192	7	177,440,512	7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	34,359,738,368	8	2,147,483,648	8	134,217,728	8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	38,654,705,664	9	2,415,919,104	9	150,994,944	9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	42,949,672,960	A	2,684,354,560	A	167,772,160	A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	47,244,640,256	B	2,952,790,016	B	184,549,376	B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	51,539,607,552	C	3,221,225,472	C	201,326,592	C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	55,834,574,848	D	3,489,660,928	D	218,103,808	D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	60,129,542,144	E	3,758,096,384	E	234,881,024	E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	64,424,509,440	F	4,026,531,840	F	251,658,240	F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
9		8		7		6		5		4		3		2		1	

HEXADECIMAL TO DECIMAL

This table allows for conversion of hexadecimal numbers of up to nine characters in length to their decimal equivalents.

Locate the columns in the table corresponding to the position of each character of the hexadecimal number. Record the decimal equivalents of the characters. The sum of these numbers is the converted number. Hexadecimal number F4D is used as an example.

Hex. Char.	Column	Decimal Equiv.
F	3	3,840
4	2	64
D	1	13
		3,917 = F4D

To convert a number without using the table:

(1) Assign the units decimal equivalent to each hexadecimal character.

(2) Starting with the decimal equivalent of the most-significant character, multiply by 16, add the decimal equivalent of the next most-significant character to the result and again multiply by 16.

(3) Repeat this process until the last character is added.

Hexadecimal number F4D is again used as the example.

```
Hex. Char.   Units Dec. Equiv.
   F                15            15
                                 X16
                                 240
   4                 4            +4
                                 244
                                 X16
                               3,904
   D                13           +13
                               3,917 = F4D
```

DECIMAL TO HEXADECIMAL

To convert decimal to hexadecimal using the table:

(1) Select the largest decimal number that is equal to or less than the number to be converted. Record the hexadecimal equivalent as the most-significant character of the hexadecimal number.

(2) Subtract the selected number from the number to be converted.

(3) Select the decimal number that is equal to or less than the result obtained from step 2 and record the hexadecimal equivalent as the second most-significant digit.

(4) Continue the process until there is no remainder.

Decimal number 3,917 is used as the example.

```
Decimal Number
from Table
   3,840             3,917
                    -3,840 --> F4D
     64                 77
                       -64
     13                 13
                       -13
```

Conversion without using the table is accomplished by successively dividing by 16 and collecting the remainders in reverse order as shown below.

```
      244
  16 |3917
      32
       71
       64
        77
        64
        13  \
             \
      15      \
  16 |244      \
      16        \
       84        \
       80         \
        4 \        \
           \        \
       0    \        \
  16 |15 ----------> F4D
```

APPENDIX D

HEXADECIMAL/DECIMAL FRACTIONS

Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
.0	.0000	.00	.0000 0000	.000	.0000 0000 0000	.0000	.0000 0000 0000 0000	.00000	.0000 0000 0000 0000 0000
.1	.0625	.01	.0039 0625	.001	.0002 4414 0625	.0001	.0000 1525 8789 0625	.00001	.0000 0095 3674 3164 0625
.2	.1250	.02	.0078 1250	.002	.0004 8828 1250	.0002	.0000 3051 7578 1250	.00002	.0000 0190 7348 6328 1250
.3	.1875	.03	.0117 1875	.003	.0007 3242 1875	.0003	.0000 4577 6367 1875	.00003	.0000 0286 1022 9492 1875
.4	.2500	.04	.0156 2500	.004	.0009 7656 2500	.0004	.0000 6103 5156 2500	.00004	.0000 0381 4697 2656 2500
.5	.3125	.05	.0195 3125	.005	.0012 2070 3125	.0005	.0000 7629 3945 3125	.00005	.0000 0476 8371 5820 3125
.6	.3750	.06	.0234 3750	.006	.0014 6484 3750	.0006	.0000 9155 2734 3750	.00006	.0000 0572 2045 8984 3750
.7	.4375	.07	.0273 4375	.007	.0017 0898 4375	.0007	.0001 0681 1523 4375	.00007	.0000 0667 5720 2148 4375
.8	.5000	.08	.0312 5000	.008	.0019 5312 5000	.0008	.0001 2207 0312 5000	.00008	.0000 0762 9394 5312 5000
.9	.5625	.09	.0351 5625	.009	.0021 9726 5625	.0009	.0001 3732 9101 5625	.00009	.0000 0858 3068 8476 5625
.A	.6250	.0A	.0390 6250	.00A	.0024 4140 6250	.000A	.0001 5258 7890 6250	.0000A	.0000 0953 6743 1640 6250
.B	.6875	.0B	.0429 6875	.00B	.0026 8554 6875	.000B	.0001 6784 6679 6875	.0000B	.0000 1049 0417 4804 6875
.C	.7500	.0C	.0468 7500	.00C	.0029 2968 7500	.000C	.0001 8310 5468 7500	.0000C	.0000 1144 4091 7968 7500
.D	.8125	.0D	.0507 8125	.00D	.0031 7382 8125	.000D	.0001 9836 4257 8125	.0000D	.0000 1239 7766 1132 8125
.E	.8750	.0E	.0546 8750	.00E	.0034 1796 8750	.000E	.0002 1362 3046 8750	.0000E	.0000 1335 1440 4296 8750
.F	.9375	.0F	.0585 9375	.00F	.0036 6210 9375	.000F	.0002 2888 1835 9375	.0000F	.0000 1430 5114 7460 9375
1		2		3		4		5	

FRACTIONAL HEXADECIMAL TO DECIMAL

When using the table, fractional hexadecimal to decimal conversion is accomplished in the same manner as for integer conversion. Hexadecimal .F4D is converted as shown below:

Hex. Char.	Column	Decimal Equiv.	
.F	1	.9375	
.04	2	.0156 2500	
.00D	3	.0031 7382 8125	
		.9562 9882 8125	= .F4D

Conversion without using the table is accomplished as follows:

.F4D = .956298828125

$$.F4D = \frac{F4D_{16}}{16^3} = \frac{3917}{4096} = .956298828125$$

FRACTIONAL DECIMAL TO HEXADECIMAL

Fractional decimal to hexadecimal conversion is accomplished in the same manner as for integer conversion when using the table. Decimal .9563 is converted as shown below.

```
 .9563
-.9375                    = .F
 .0188  0000
-.0156  2500              = .04
 .0031  7500  0000
-.0031  7382  8125        = .00D
 .0000  0117  1875          .F4D
```

Conversion without using the table is accomplished by multiplying successively by 16 and collecting the integers from the products.

```
        .9563
          X16
      15.3008
          X16
       4.8128
          X16
      13.0048
.F4D
```

APPENDIX E

TABLE OF POWERS OF SIXTEEN

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

APPENDIX F

CONVERSION TABLES

This appendix contains the following reference tables:

Title

Hexadecimal Arithmetic

- Addition Table
- Multiplication Table
- Powers of 16_{10}
- Powers of 10_{16}

Hexadecimal-Decimal Integer Conversion

Hexadecimal-Decimal Fraction Conversion

Powers of Two

Mathematical Constants

HEXADECIMAL ARITHMETIC

ADDITION TABLE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	1E	2B	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

TABLE OF POWERS OF SIXTEEN$_{10}$

16^n	n	16^{-n}
1	0	0.10000 00000 00000 00000 × 10
16	1	0.62500 00000 00000 00000 × 10^{-1}
256	2	0.39062 50000 00000 00000 × 10^{-2}
4 096	3	0.24414 06250 00000 00000 × 10^{-3}
65 536	4	0.15258 78906 25000 00000 × 10^{-4}
1 048 576	5	0.95367 43164 06250 00000 × 10^{-6}
16 777 216	6	0.59604 64477 53906 25000 × 10^{-7}
268 435 456	7	0.37252 90298 46191 40625 × 10^{-8}
4 294 967 296	8	0.23283 06436 53869 62891 × 10^{-9}
68 719 476 736	9	0.14551 91522 83668 51807 × 10^{-10}
1 099 511 627 776	10	0.90949 47017 72928 23792 × 10^{-12}
17 592 186 044 416	11	0.56843 41886 08080 14870 × 10^{-13}
281 474 976 710 656	12	0.35527 13678 80050 09294 × 10^{-14}
4 503 599 627 370 496	13	0.22204 46049 25031 30808 × 10^{-15}
72 057 594 037 927 936	14	0.13877 78780 78144 56755 × 10^{-16}
1 152 921 504 606 846 976	15	0.86736 17379 88403 54721 × 10^{-18}

TABLE OF POWERS OF 10$_{16}$

10^n	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C28F 5C28 F5C3 × 16^{-1}
3E8	3	0.4189 374B C6A7 EF9E × 16^{-2}
2710	4	0.68DB 8BAC 710C B296 × 16^{-3}
1 86A0	5	0.A7C5 AC47 1B47 8423 × 16^{-4}
F 4240	6	0.10C6 F7A0 B5ED 8D37 × 16^{-4}
98 9680	7	0.1AD7 F29A BCAF 4858 × 16^{-5}
5F5 E100	8	0.2AF3 1DC4 6118 73BF × 16^{-6}
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC × 16^{-7}
2 540B E400	10	0.6DF3 7F67 5EF6 EADF × 16^{-8}
17 4876 E800	11	0.AFEB FF0B CB24 AAFF × 16^{-9}
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 × 16^{-9}
918 4E72 A000	13	0.1C25 C268 4976 81C2 × 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604 × 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D × 16^{-12}
23 8652 6FC1 0000	16	0.734A CA5F 6226 F0AE × 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 × 16^{-14}
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 × 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 × 16^{-15}

HEXADECIMAL-DECIMAL INTEGER CONVERSION

The table below provides for direct conversions between hexadecimal integers in the range 0–FFF and decimal integers in the range 0–4095. For conversion of larger integers, the table values may be added to the following figures:

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

Hexadecimal fractions may be converted to decimal fractions as follows:

1. Express the hexadecimal fraction as an integer times 16^{-n}, where n is the number of significant hexadecimal places to the right of the hexadecimal point.

$$0.\,CA9BF3_{16} = CA9\ BF3_{16} \times 16^{-6}$$

2. Find the decimal equivalent of the hexadecimal integer

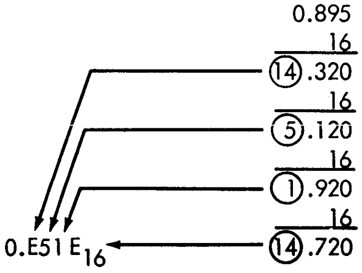
$$CA9\ BF3_{16} = 13\ 278\ 195_{10}$$

3. Multiply the decimal equivalent by 16^{-n}

$$\begin{array}{r} 13\ 278\ 195 \\ \underline{\times\ 596\ 046\ 448 \times 10^{-16}} \\ 0.791\ 442\ 096_{10} \end{array}$$

Decimal fractions may be converted to hexadecimal fractions by successively multiplying the decimal fraction by 16_{10}. After each multiplication, the integer portion is removed to form a hexadecimal fraction by building to the right of the hexadecimal point. However, since decimal arithmetic is used in this conversion, the integer portion of each product must be converted to hexadecimal numbers.

Example: Convert 0.895_{10} to its hexadecimal equivalent



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

HEXADECIMAL-DECIMAL INTEGER CONVERSION (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

HEXADECIMAL-DECIMAL INTEGER CONVERSION (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

HEXADECIMAL-DECIMAL INTEGER CONVERSION (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

HEXADECIMAL-DECIMAL INTEGER CONVERSION (cont.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	376)	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

HEXADECIMAL-DECIMAL FRACTION CONVERSION

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.40 00 00 00	.25000 00000	.80 00 00 00	.50000 00000	.C0 00 00 00	.75000 00000
.01 00 00 00	.00390 62500	.41 00 00 00	.25390 62500	.81 00 00 00	.50390 62500	.C1 00 00 00	.75390 62500
.02 00 00 00	.00781 25000	.42 00 00 00	.25781 25000	.82 00 00 00	.50781 25000	.C2 00 00 00	.75781 25000
.03 00 00 00	.01171 87500	.43 00 00 00	.26171 87500	.83 00 00 00	.51171 87500	.C3 00 00 00	.76171 87500
.04 00 00 00	.01562 50000	.44 00 00 00	.26562 50000	.84 00 00 00	.51562 50000	.C4 00 00 00	.76562 50000
.05 00 00 00	.01953 12500	.45 00 00 00	.26953 12500	.85 00 00 00	.51953 12500	.C5 00 00 00	.76953 12500
.06 00 00 00	.02343 75000	.46 00 00 00	.27343 75000	.86 00 00 00	.52343 75000	.C6 00 00 00	.77343 75000
.07 00 00 00	.02734 37500	.47 00 00 00	.27734 37500	.87 00 00 00	.52734 37500	.C7 00 00 00	.77734 37500
.08 00 00 00	.03125 00000	.48 00 00 00	.28125 00000	.88 00 00 00	.53125 00000	.C8 00 00 00	.78125 00000
.09 00 00 00	.03515 62500	.49 00 00 00	.28515 62500	.89 00 00 00	.53515 62500	.C9 00 00 00	.78515 62500
.0A 00 00 00	.03906 25000	.4A 00 00 00	.28906 25000	.8A 00 00 00	.53906 25000	.CA 00 00 00	.78906 25000
.0B 00 00 00	.04296 87500	.4B 00 00 00	.29296 87500	.8B 00 00 00	.54296 87500	.CB 00 00 00	.79296 87500
.0C 00 00 00	.04687 50000	.4C 00 00 00	.29687 50000	.8C 00 00 00	.54687 50000	.CC 00 00 00	.79687 50000
.0D 00 00 00	.05078 12500	.4D 00 00 00	.30078 12500	.8D 00 00 00	.55078 12500	.CD 00 00 00	.80078 12500
.0E 00 00 00	.05468 75000	.4E 00 00 00	.30468 75000	.8E 00 00 00	.55468 75000	.CE 00 00 00	.80468 75000
.0F 00 00 00	.05859 37500	.4F 00 00 00	.30859 37500	.8F 00 00 00	.55859 37500	.CF 00 00 00	.80859 37500
.10 00 00 00	.06250 00000	.50 00 00 00	.31250 00000	.90 00 00 00	.56250 00000	.D0 00 00 00	.81250 00000
.11 00 00 00	.06640 62500	.51 00 00 00	.31640 62500	.91 00 00 00	.56640 62500	.D1 00 00 00	.81640 62500
.12 00 00 00	.07031 25000	.52 00 00 00	.32031 25000	.92 00 00 00	.57031 25000	.D2 00 00 00	.82031 25000
.13 00 00 00	.07421 87500	.53 00 00 00	.32421 87500	.93 00 00 00	.57421 87500	.D3 00 00 00	.82421 87500
.14 00 00 00	.07812 50000	.54 00 00 00	.32812 50000	.94 00 00 00	.57812 50000	.D4 00 00 00	.82812 50000
.15 00 00 00	.08203 12500	.55 00 00 00	.33203 12500	.95 00 00 00	.58203 12500	.D5 00 00 00	.83203 12500
.16 00 00 00	.08593 75000	.56 00 00 00	.33593 75000	.96 00 00 00	.58593 75000	.D6 00 00 00	.83593 75000
.17 00 00 00	.08984 37500	.57 00 00 00	.33984 37500	.97 00 00 00	.58984 37500	.D7 00 00 00	.83984 37500
.18 00 00 00	.09375 00000	.58 00 00 00	.34375 00000	.98 00 00 00	.59375 00000	.D8 00 00 00	.84375 00000
.19 00 00 00	.09765 62500	.59 00 00 00	.34765 62500	.99 00 00 00	.59765 62500	.D9 00 00 00	.84765 62500
.1A 00 00 00	.10156 25000	.5A 00 00 00	.35156 25000	.9A 00 00 00	.60156 25000	.DA 00 00 00	.85156 25000
.1B 00 00 00	.10546 87500	.5B 00 00 00	.35546 87500	.9B 00 00 00	.60546 87500	.DB 00 00 00	.85546 87500
.1C 00 00 00	.10937 50000	.5C 00 00 00	.35937 50000	.9C 00 00 00	.60937 50000	.DC 00 00 00	.85937 50000
.1D 00 00 00	.11328 12500	.5D 00 00 00	.36328 12500	.9D 00 00 00	.61328 12500	.DD 00 00 00	.86328 12500
.1E 00 00 00	.11718 75000	.5E 00 00 00	.36718 75000	.9E 00 00 00	.61718 75000	.DE 00 00 00	.86718 75000
.1F 00 00 00	.12109 37500	.5F 00 00 00	.37109 37500	.9F 00 00 00	.62109 37500	.DF 00 00 00	.87109 37500
.20 00 00 00	.12500 00000	.60 00 00 00	.37500 00000	.A0 00 00 00	.62500 00000	.E0 00 00 00	.87500 00000
.21 00 00 00	.12890 62500	.61 00 00 00	.37890 62500	.A1 00 00 00	.62890 62500	.E1 00 00 00	.87890 62500
.22 00 00 00	.13281 25000	.62 00 00 00	.38281 25000	.A2 00 00 00	.63281 25000	.E2 00 00 00	.88281 25000
.23 00 00 00	.13671 87500	.63 00 00 00	.38671 87500	.A3 00 00 00	.63671 87500	.E3 00 00 00	.88671 87500
.24 00 00 00	.14062 50000	.64 00 00 00	.39062 50000	.A4 00 00 00	.64062 50000	.E4 00 00 00	.89062 50000
.25 00 00 00	.14453 12500	.65 00 00 00	.39453 12500	.A5 00 00 00	.64453 12500	.E5 00 00 00	.89453 12500
.26 00 00 00	.14843 75000	.66 00 00 00	.39843 75000	.A6 00 00 00	.64843 75000	.E6 00 00 00	.89843 75000
.27 00 00 00	.15234 37500	.67 00 00 00	.40234 37500	.A7 00 00 00	.65234 37500	.E7 00 00 00	.90234 37500
.28 00 00 00	.15625 00000	.68 00 00 00	.40625 00000	.A8 00 00 00	.65625 00000	.E8 00 00 00	.90625 00000
.29 00 00 00	.16015 62500	.69 00 00 00	.41015 62500	.A9 00 00 00	.66015 62500	.E9 00 00 00	.91015 62500
.2A 00 00 00	.16406 25000	.6A 00 00 00	.41406 25000	.AA 00 00 00	.66406 25000	.EA 00 00 00	.91406 25000
.2B 00 00 00	.16796 87500	.6B 00 00 00	.41796 87500	.AB 00 00 00	.66796 87500	.EB 00 00 00	.91796 87500
.2C 00 00 00	.17187 50000	.6C 00 00 00	.42187 50000	.AC 00 00 00	.67187 50000	.EC 00 00 00	.92187 50000
.2D 00 00 00	.17578 12500	.6D 00 00 00	.42578 12500	.AD 00 00 00	.67578 12500	.ED 00 00 00	.92578 12500
.2E 00 00 00	.17968 75000	.6E 00 00 00	.42968 75000	.AE 00 00 00	.67968 75000	.EE 00 00 00	.92968 75000
.2F 00 00 00	.18359 37500	.6F 00 00 00	.43359 37500	.AF 00 00 00	.68359 37500	.EF 00 00 00	.93359 37500
.30 00 00 00	.18750 00000	.70 00 00 00	.43750 00000	.B0 00 00 00	.68750 00000	.F0 00 00 00	.93750 00000
.31 00 00 00	.19140 62500	.71 00 00 00	.44140 62500	.B1 00 00 00	.69140 62500	.F1 00 00 00	.94140 62500
.32 00 00 00	.19531 25000	.72 00 00 00	.44531 25000	.B2 00 00 00	.69531 25000	.F2 00 00 00	.94531 25000
.33 00 00 00	.19921 87500	.73 00 00 00	.44921 87500	.B3 00 00 00	.69921 87500	.F3 00 00 00	.94921 87500
.34 00 00 00	.20312 50000	.74 00 00 00	.45312 50000	.B4 00 00 00	.70312 50000	.F4 00 00 00	.95312 50000
.35 00 00 00	.20703 12500	.75 00 00 00	.45703 12500	.B5 00 00 00	.70703 12500	.F5 00 00 00	.95703 12500
.36 00 00 00	.21093 75000	.76 00 00 00	.46093 75000	.B6 00 00 00	.71093 75000	.F6 00 00 00	.96093 75000
.37 00 00 00	.21484 37500	.77 00 00 00	.46484 37500	.B7 00 00 00	.71484 37500	.F7 00 00 00	.96484 37500
.38 00 00 00	.21875 00000	.78 00 00 00	.46875 00000	.B8 00 00 00	.71875 00000	.F8 00 00 00	.96875 00000
.39 00 00 00	.22265 62500	.79 00 00 00	.47265 62500	.B9 00 00 00	.72265 62500	.F9 00 00 00	.97265 62500
.3A 00 00 00	.22656 25000	.7A 00 00 00	.47656 25000	.BA 00 00 00	.72656 25000	.FA 00 00 00	.97656 25000
.3B 00 00 00	.23046 87500	.7B 00 00 00	.48046 87500	.BB 00 00 00	.73046 87500	.FB 00 00 00	.98046 87500
.3C 00 00 00	.23437 50000	.7C 00 00 00	.48437 50000	.BC 00 00 00	.73437 50000	.FC 00 00 00	.98437 50000
.3D 00 00 00	.23828 12500	.7D 00 00 00	.48828 12500	.BD 00 00 00	.73828 12500	.FD 00 00 00	.98828 12500
.3E 00 00 00	.24218 75000	.7E 00 00 00	.49218 75000	.BE 00 00 00	.74218 75000	.FE 00 00 00	.99218 75000
.3F 00 00 00	.24609 37500	.7F 00 00 00	.49609 37500	.BF 00 00 00	.74609 37500	.FF 00 00 00	.99609 37500

HEXADECIMAL-DECIMAL FRACTION CONVERSION (cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 40 00 00	.00097 65625	.00 80 00 00	.00195 31250	.00 C0 00 00	.00292 96875
.00 01 00 00	.00001 52587	.00 41 00 00	.00099 18212	.00 81 00 00	.00196 83837	.00 C1 00 00	.00294 49462
.00 02 00 00	.00003 05175	.00 42 00 00	.00100 70800	.00 82 00 00	.00198 36425	.00 C2 00 00	.00296 02050
.00 03 00 00	.00004 57763	.00 43 00 00	.00102 23388	.00 83 00 00	.00199 89013	.00 C3 00 00	.00297 54638
.00 04 00 00	.00006 10351	.00 44 00 00	.00103 75976	.00 84 00 00	.00201 41601	.00 C4 00 00	.00299 07226
.00 05 00 00	.00007 62939	.00 45 00 00	.00105 28564	.00 85 00 00	.00202 94189	.00 C5 00 00	.00300 59814
.00 06 00 00	.00009 15527	.00 46 00 00	.00106 81152	.00 86 00 00	.00204 46777	.00 C6 00 00	.00302 12402
.00 07 00 00	.00010 68115	.00 47 00 00	.00108 33740	.00 87 00 00	.00205 99365	.00 C7 00 00	.00303 64990
.00 08 00 00	.00012 20703	.00 48 00 00	.00109 86328	.00 88 00 00	.00207 51953	.00 C8 00 00	.00305 17578
.00 09 00 00	.00013 73291	.00 49 00 00	.00111 38916	.00 89 00 00	.00209 04541	.00 C9 00 00	.00306 70166
.00 0A 00 00	.00015 25878	.00 4A 00 00	.00112 91503	.00 8A 00 00	.00210 57128	.00 CA 00 00	.00308 22753
.00 0B 00 00	.00016 78466	.00 4B 00 00	.00114 44091	.00 8B 00 00	.00212 09716	.00 CB 00 00	.00309 75341
.00 0C 00 00	.00018 31054	.00 4C 00 00	.00115 96679	.00 8C 00 00	.00213 62304	.00 CC 00 00	.00311 27929
.00 0D 00 00	.00019 83642	.00 4D 00 00	.00117 49267	.00 8D 00 00	.00215 14892	.00 CD 00 00	.00312 80517
.00 0E 00 00	.00021 36230	.00 4E 00 00	.00119 01855	.00 8E 00 00	.00216 67480	.00 CE 00 00	.00314 33105
.00 0F 00 00	.00022 88818	.00 4F 00 00	.00120 54443	.00 8F 00 00	.00218 20068	.00 CF 00 00	.00315 85693
.00 10 00 00	.00024 41406	.00 50 00 00	.00122 07031	.00 90 00 00	.00219 72656	.00 D0 00 00	.00317 38281
.00 11 00 00	.00025 93994	.00 51 00 00	.00123 59619	.00 91 00 00	.00221 25244	.00 D1 00 00	.00318 90869
.00 12 00 00	.00027 46582	.00 52 00 00	.00125 12207	.00 92 00 00	.00222 77832	.00 D2 00 00	.00320 43457
.00 13 00 00	.00028 99169	.00 53 00 00	.00126 64794	.00 93 00 00	.00224 30419	.00 D3 00 00	.00321 96044
.00 14 00 00	.00030 51757	.00 54 00 00	.00128 17382	.00 94 00 00	.00225 83007	.00 D4 00 00	.00323 48632
.00 15 00 00	.00032 04345	.00 55 00 00	.00129 69970	.00 95 00 00	.00227 35595	.00 D5 00 00	.00325 01220
.00 16 00 00	.00033 56933	.00 56 00 00	.00131 22558	.00 96 00 00	.00228 88183	.00 D6 00 00	.00326 53808
.00 17 00 00	.00035 09521	.00 57 00 00	.00132 75146	.00 97 00 00	.00230 40771	.00 D7 00 00	.00328 06396
.00 18 00 00	.00036 62109	.00 58 00 00	.00134 27734	.00 98 00 00	.00231 93359	.00 D8 00 00	.00329 58984
.00 19 00 00	.00038 14697	.00 59 00 00	.00135 80322	.00 99 00 00	.00233 45947	.00 D9 00 00	.00331 11572
.00 1A 00 00	.00039 67285	.00 5A 00 00	.00137 32910	.00 9A 00 00	.00234 98535	.00 DA 00 00	.00332 64160
.00 1B 00 00	.00041 19873	.00 5B 00 00	.00138 85498	.00 9B 00 00	.00236 51123	.00 DB 00 00	.00334 16748
.00 1C 00 00	.00042 72460	.00 5C 00 00	.00140 38085	.00 9C 00 00	.00238 03710	.00 DC 00 00	.00335 69335
.00 1D 00 00	.00044 25048	.00 5D 00 00	.00141 90673	.00 9D 00 00	.00239 56298	.00 DD 00 00	.00337 21923
.00 1E 00 00	.00045 77636	.00 5E 00 00	.00143 43261	.00 9E 00 00	.00241 08886	.00 DE 00 00	.00338 74511
.00 1F 00 00	.00047 30224	.00 5F 00 00	.00144 95849	.00 9F 00 00	.00242 61474	.00 DF 00 00	.00340 27099
.00 20 00 00	.00048 82812	.00 60 00 00	.00146 48437	.00 A0 00 00	.00244 14062	.00 E0 00 00	.00341 79687
.00 21 00 00	.00050 35400	.00 61 00 00	.00148 01025	.00 A1 00 00	.00245 66650	.00 E1 00 00	.00343 32275
.00 22 00 00	.00051 87988	.00 62 00 00	.00149 53613	.00 A2 00 00	.00247 19238	.00 E2 00 00	.00344 84863
.00 23 00 00	.00053 40576	.00 63 00 00	.00151 06201	.00 A3 00 00	.00248 71826	.00 E3 00 00	.00346 37451
.00 24 00 00	.00054 93164	.00 64 00 00	.00152 58789	.00 A4 00 00	.00250 24414	.00 E4 00 00	.00347 90039
.00 25 00 00	.00056 45751	.00 65 00 00	.00154 11376	.00 A5 00 00	.00251 77001	.00 E5 00 00	.00349 42626
.00 26 00 00	.00057 98339	.00 66 00 00	.00155 63964	.00 A6 00 00	.00253 29589	.00 E6 00 00	.00350 95214
.00 27 00 00	.00059 50927	.00 67 00 00	.00157 16552	.00 A7 00 00	.00254 82177	.00 E7 00 00	.00352 47802
.00 28 00 00	.00061 03515	.00 68 00 00	.00158 69140	.00 A8 00 00	.00256 34765	.00 E8 00 00	.00354 00390
.00 29 00 00	.00062 56103	.00 69 00 00	.00160 21728	.00 A9 00 00	.00257 87353	.00 E9 00 00	.00355 52978
.00 2A 00 00	.00064 08691	.00 6A 00 00	.00161 74316	.00 AA 00 00	.00259 39941	.00 EA 00 00	.00357 05566
.00 2B 00 00	.00065 61279	.00 6B 00 00	.00163 26904	.00 AB 00 00	.00260 92529	.00 EB 00 00	.00358 58154
.00 2C 00 00	.00067 13867	.00 6C 00 00	.00164 79492	.00 AC 00 00	.00262 45117	.00 EC 00 00	.00360 10742
.00 2D 00 00	.00068 66455	.00 6D 00 00	.00166 32080	.00 AD 00 00	.00263 97705	.00 ED 00 00	.00361 63330
.00 2E 00 00	.00070 19042	.00 6E 00 00	.00167 84667	.00 AE 00 00	.00265 50292	.00 EE 00 00	.00363 15917
.00 2F 00 00	.00071 71630	.00 6F 00 00	.00169 37255	.00 AF 00 00	.00267 02880	.00 EF 00 00	.00364 68505
.00 30 00 00	.00073 24218	.00 70 00 00	.00170 89843	.00 B0 00 00	.00268 55468	.00 F0 00 00	.00366 21093
.00 31 00 00	.00074 76806	.00 71 00 00	.00172 42431	.00 B1 00 00	.00270 08056	.00 F1 00 00	.00367 73681
.00 32 00 00	.00076 29394	.00 72 00 00	.00173 95019	.00 B2 00 00	.00271 60644	.00 F2 00 00	.00369 26269
.00 33 00 00	.00077 81982	.00 73 00 00	.00175 47607	.00 B3 00 00	.00273 13232	.00 F3 00 00	.00370 78857
.00 34 00 00	.00079 34570	.00 74 00 00	.00177 00195	.00 B4 00 00	.00274 65820	.00 F4 00 00	.00372 31445
.00 35 00 00	.00080 87158	.00 75 00 00	.00178 52783	.00 B5 00 00	.00276 18408	.00 F5 00 00	.00373 84033
.00 36 00 00	.00082 39746	.00 76 00 00	.00180 05371	.00 B6 00 00	.00277 70996	.00 F6 00 00	.00375 36621
.00 37 00 00	.00083 92333	.00 77 00 00	.00181 57958	.00 B7 00 00	.00279 23583	.00 F7 00 00	.00376 89208
.00 38 00 00	.00085 44921	.00 78 00 00	.00183 10546	.00 B8 00 00	.00280 76171	.00 F8 00 00	.00378 41796
.00 39 00 00	.00086 97509	.00 79 00 00	.00184 63134	.00 B9 00 00	.00282 28759	.00 F9 00 00	.00379 94384
.00 3A 00 00	.00088 50097	.00 7A 00 00	.00186 15722	.00 BA 00 00	.00283 81347	.00 FA 00 00	.00381 46972
.00 3B 00 00	.00090 02685	.00 7B 00 00	.00187 68310	.00 BB 00 00	.00285 33935	.00 FB 00 00	.00382 99560
.00 3C 00 00	.00091 55273	.00 7C 00 00	.00189 20898	.00 BC 00 00	.00286 86523	.00 FC 00 00	.00384 52148
.00 3D 00 00	.00093 07861	.00 7D 00 00	.00190 73486	.00 BD 00 00	.00288 39111	.00 FD 00 00	.00386 04736
.00 3E 00 00	.00094 60449	.00 7E 00 00	.00192 26074	.00 BE 00 00	.00289 91699	.00 FE 00 00	.00387 57324
.00 3F 00 00	.00096 13037	.00 7F 00 00	.00193 78662	.00 BF 00 00	.00291 44287	.00 FF 00 00	.00389 09912

HEXADECIMAL-DECIMAL FRACTION CONVERSION (cont.)

Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal
.00 00 00 00	.00000 00000	.00 00 40 00	.00000 38146	.00 00 80 00	.00000 76293	.00 00 C0 00	.00001 14440
.00 00 01 00	.00000 00596	.00 00 41 00	.00000 38743	.00 00 81 00	.00000 76889	.00 00 C1 00	.00001 15036
.00 00 02 00	.00000 01192	.00 00 42 00	.00000 39339	.00 00 82 00	.00000 77486	.00 00 C2 00	.00001 15633
.00 00 03 00	.00000 01788	.00 00 43 00	.00000 39935	.00 00 83 00	.00000 78082	.00 00 C3 00	.00001 16229
.00 00 04 00	.00000 02384	.00 00 44 00	.00000 40531	.00 00 84 00	.00000 78678	.00 00 C4 00	.00001 16825
.00 00 05 00	.00000 02980	.00 00 45 00	.00000 41127	.00 00 85 00	.00000 79274	.00 00 C5 00	.00001 17421
.00 00 06 00	.00000 03576	.00 00 46 00	.00000 41723	.00 00 86 00	.00000 79870	.00 00 C6 00	.00001 18017
.00 00 07 00	.00000 04172	.00 00 47 00	.00000 42319	.00 00 87 00	.00000 80466	.00 00 C7 00	.00001 18613
.00 00 08 00	.00000 04768	.00 00 48 00	.00000 42915	.00 00 88 00	.00000 81062	.00 00 C8 00	.00001 19209
.00 00 09 00	.00000 05364	.00 00 49 00	.00000 43511	.00 00 89 00	.00000 81658	.00 00 C9 00	.00001 19805
.00 00 0A 00	.00000 05960	.00 00 4A 00	.00000 44107	.00 00 8A 00	.00000 82254	.00 00 CA 00	.00001 20401
.00 00 0B 00	.00000 06556	.00 00 4B 00	.00000 44703	.00 00 8B 00	.00000 82850	.00 00 CB 00	.00001 20997
.00 00 0C 00	.00000 07152	.00 00 4C 00	.00000 45299	.00 00 8C 00	.00000 83446	.00 00 CC 00	.00001 21593
.00 00 0D 00	.00000 07748	.00 00 4D 00	.00000 45895	.00 00 8D 00	.00000 84042	.00 00 CD 00	.00001 22189
.00 00 0E 00	.00000 08344	.00 00 4E 00	.00000 46491	.00 00 8E 00	.00000 84638	.00 00 CE 00	.00001 22785
.00 00 0F 00	.00000 08940	.00 00 4F 00	.00000 47087	.00 00 8F 00	.00000 85234	.00 00 CF 00	.00001 23381
.00 00 10 00	.00000 09536	.00 00 50 00	.00000 47683	.00 00 90 00	.00000 85830	.00 00 D0 00	.00001 23977
.00 00 11 00	.00000 10132	.00 00 51 00	.00000 48279	.00 00 91 00	.00000 86426	.00 00 D1 00	.00001 24573
.00 00 12 00	.00000 10728	.00 00 52 00	.00000 48875	.00 00 92 00	.00000 87022	.00 00 D2 00	.00001 25169
.00 00 13 00	.00000 11324	.00 00 53 00	.00000 49471	.00 00 93 00	.00000 87618	.00 00 D3 00	.00001 25765
.00 00 14 00	.00000 11920	.00 00 54 00	.00000 50067	.00 00 94 00	.00000 88214	.00 00 D4 00	.00001 26361
.00 00 15 00	.00000 12516	.00 00 55 00	.00000 50663	.00 00 95 00	.00000 88810	.00 00 D5 00	.00001 26957
.00 00 16 00	.00000 13113	.00 00 56 00	.00000 51259	.00 00 96 00	.00000 89406	.00 00 D6 00	.00001 27553
.00 00 17 00	.00000 13709	.00 00 57 00	.00000 51856	.00 00 97 00	.00000 90003	.00 00 D7 00	.00001 28149
.00 00 18 00	.00000 14305	.00 00 58 00	.00000 52452	.00 00 98 00	.00000 90599	.00 00 D8 00	.00001 28746
.00 00 19 00	.00000 14901	.00 00 59 00	.00000 53048	.00 00 99 00	.00000 91195	.00 00 D9 00	.00001 29342
.00 00 1A 00	.00000 15497	.00 00 5A 00	.00000 53644	.00 00 9A 00	.00000 91791	.00 00 DA 00	.00001 29938
.00 00 1B 00	.00000 16093	.00 00 5B 00	.00000 54240	.00 00 9B 00	.00000 92387	.00 00 DB 00	.00001 30534
.00 00 1C 00	.00000 16689	.00 00 5C 00	.00000 54836	.00 00 9C 00	.00000 92983	.00 00 DC 00	.00001 31130
.00 00 1D 00	.00000 17285	.00 00 5D 00	.00000 55432	.00 00 9D 00	.00000 93579	.00 00 DD 00	.00001 31726
.00 00 1E 00	.00000 17881	.00 00 5E 00	.00000 56028	.00 00 9E 00	.00000 94175	.00 00 DE 00	.00001 32322
.00 00 1F 00	.00000 18477	.00 00 5F 00	.00000 56624	.00 00 9F 00	.00000 94771	.00 00 DF 00	.00001 32918
.00 00 20 00	.00000 19073	.00 00 60 00	.00000 57220	.00 00 A0 00	.00000 95367	.00 00 E0 00	.00001 33514
.00 00 21 00	.00000 19669	.00 00 61 00	.00000 57816	.00 00 A1 00	.00000 95963	.00 00 E1 00	.00001 34110
.00 00 22 00	.00000 20265	.00 00 62 00	.00000 58412	.00 00 A2 00	.00000 96559	.00 00 E2 00	.00001 34706
.00 00 23 00	.00000 20861	.00 00 63 00	.00000 59008	.00 00 A3 00	.00000 97155	.00 00 E3 00	.00001 35302
.00 00 24 00	.00000 21457	.00 00 64 00	.00000 59604	.00 00 A4 00	.00000 97751	.00 00 E4 00	.00001 35898
.00 00 25 00	.00000 22053	.00 00 65 00	.00000 60200	.00 00 A5 00	.00000 98347	.00 00 E5 00	.00001 36494
.00 00 26 00	.00000 22649	.00 00 66 00	.00000 60796	.00 00 A6 00	.00000 98943	.00 00 E6 00	.00001 37090
.00 00 27 00	.00000 23245	.00 00 67 00	.00000 61392	.00 00 A7 00	.00000 99539	.00 00 E7 00	.00001 37686
.00 00 28 00	.00000 23841	.00 00 68 00	.00000 61988	.00 00 A8 00	.00001 00135	.00 00 E8 00	.00001 38282
.00 00 29 00	.00000 24437	.00 00 69 00	.00000 62584	.00 00 A9 00	.00001 00731	.00 00 E9 00	.00001 38878
.00 00 2A 00	.00000 25033	.00 00 6A 00	.00000 63180	.00 00 AA 00	.00001 01327	.00 00 EA 00	.00001 39474
.00 00 2B 00	.00000 25629	.00 00 6B 00	.00000 63776	.00 00 AB 00	.00001 01923	.00 00 EB 00	.00001 40070
.00 00 2C 00	.00000 26226	.00 00 6C 00	.00000 64373	.00 00 AC 00	.00001 02519	.00 00 EC 00	.00001 40666
.00 00 2D 00	.00000 26822	.00 00 6D 00	.00000 64969	.00 00 AD 00	.00001 03116	.00 00 ED 00	.00001 41263
.00 00 2E 00	.00000 27418	.00 00 6E 00	.00000 65565	.00 00 AE 00	.00001 03712	.00 00 EE 00	.00001 41859
.00 00 2F 00	.00000 28014	.00 00 6F 00	.00000 66161	.00 00 AF 00	.00001 04308	.00 00 EF 00	.00001 42455
.00 00 30 00	.00000 28610	.00 00 70 00	.00000 66757	.00 00 B0 00	.00001 04904	.00 00 F0 00	.00001 43051
.00 00 31 00	.00000 29206	.00 00 71 00	.00000 67353	.00 00 B1 00	.00001 05500	.00 00 F1 00	.00001 43647
.00 00 32 00	.00000 29802	.00 00 72 00	.00000 67949	.00 00 B2 00	.00001 06096	.00 00 F2 00	.00001 44243
.00 00 33 00	.00000 30398	.00 00 73 00	.00000 68545	.00 00 B3 00	.00001 06692	.00 00 F3 00	.00001 44839
.00 00 34 00	.00000 30994	.00 00 74 00	.00000 69141	.00 00 B4 00	.00001 07288	.00 00 F4 00	.00001 45435
.00 00 35 00	.00000 31590	.00 00 75 00	.00000 69737	.00 00 B5 00	.00001 07884	.00 00 F5 00	.00001 46031
.00 00 36 00	.00000 32186	.00 00 76 00	.00000 70333	.00 00 B6 00	.00001 08480	.00 00 F6 00	.00001 46627
.00 00 37 00	.00000 32782	.00 00 77 00	.00000 70929	.00 00 B7 00	.00001 09076	.00 00 F7 00	.00001 47223
.00 00 38 00	.00000 33378	.00 00 78 00	.00000 71525	.00 00 B8 00	.00001 09672	.00 00 F8 00	.00001 47819
.00 00 39 00	.00000 33974	.00 00 79 00	.00000 72121	.00 00 B9 00	.00001 10268	.00 00 F9 00	.00001 48415
.00 00 3A 00	.00000 34570	.00 00 7A 00	.00000 72717	.00 00 BA 00	.00001 10864	.00 00 FA 00	.00001 49011
.00 00 3B 00	.00000 35166	.00 00 7B 00	.00000 73313	.00 00 BB 00	.00001 11460	.00 00 FB 00	.00001 49607
.00 00 3C 00	.00000 35762	.00 00 7C 00	.00000 73909	.00 00 BC 00	.00001 12056	.00 00 FC 00	.00001 50203
.00 00 3D 00	.00000 36358	.00 00 7D 00	.00000 74505	.00 00 BD 00	.00001 12652	.00 00 FD 00	.00001 50799
.00 00 3E 00	.00000 36954	.00 00 7E 00	.00000 75101	.00 00 BE 00	.00001 13248	.00 00 FE 00	.00001 51395
.00 00 3F 00	.00000 37550	.00 00 7F 00	.00000 75697	.00 00 BF 00	.00001 13844	.00 00 FF 00	.00001 51991