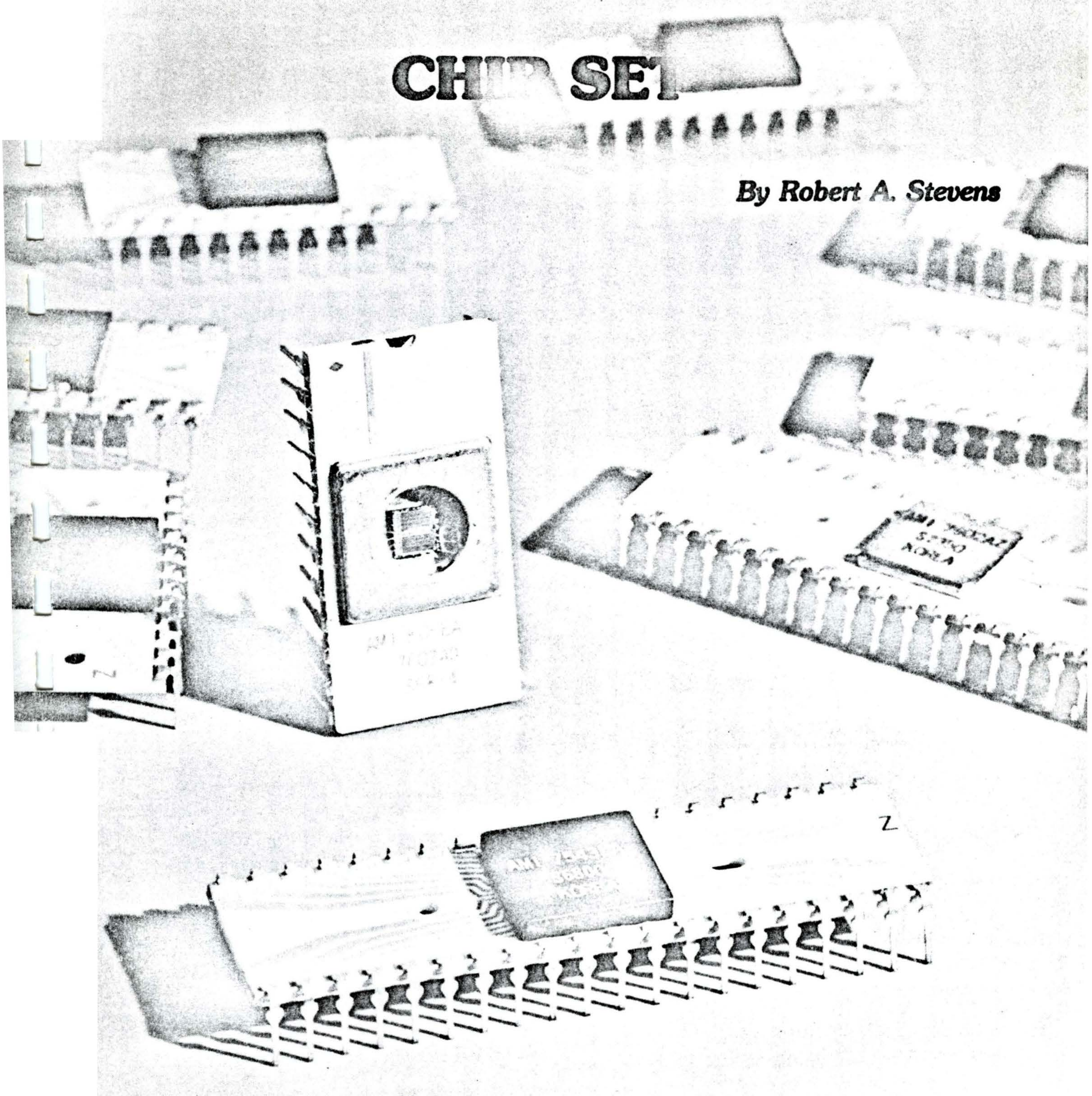
AMI 6800 MICROCOMPUTER CHIP SET

By Robert A. Stevens



This article is the first one of a series of four articles covering AMI's S6800 microcomputer chip set, EVK Microcomputer Prototyping boards and EVK prototyping board PROTO & (RS)^3 program development software.

The first article in this series covers the S6800 MPU in detail and summarizes the microcomputer supporting IC's in order to lay the ground work for next months article on AMI's EVK Prototyping boards.

AMI's S6800 FAMILY OF MICROCOMPUTER IC's

AMI's S6800 family of microcomputer, IC's is composed of a series of matched MOS large scale integrated (LSI) circuits for, configuring into microcomputer systems. This series of microcomputer MOS LSI functional building block logic circuits include a MPU, ROM, EPROM, RAM, PIA, ACIA, USRT, and Digital Modem Logic circuits.

S6800 — 8-BIT MICROPROCESSOR

FUNCTIONAL DESCRIPTION

S6800 MICROPROCESSOR (MPU) — an 8-bit parallel processor, with the ability to address up to 65K bytes of memory, and execute instructions in 2 microseconds. It is manufactured using N-channel MOS technology and operates on a single +5V power supply. All inputs and outputs are TTL compatible. The MPU has six internal registers, four types of vectored interrupts and 72 basic instructions. The basic instructions can be used in different addressing modes to save instruction execution time and memory space.

FEATURES

- Eight-Bit Parallel Processing
- Bi-Directional Data Bus
- Sixteen-Bit Address Bus — 65536 Bytes of Addressing
- 72 Instructions — Variable Length
- Seven Addressing Modes — Direct, Relative, Immediate, Indexed, Extended, Implied and Accumulator
- Variable Length Stack
- Vectored Restart
- 2 Microsecond Instruction Execution
- Maskable Interrupt Vector
- Separate Non-Maskable Interrupt — Internal Registers Saved in Stack

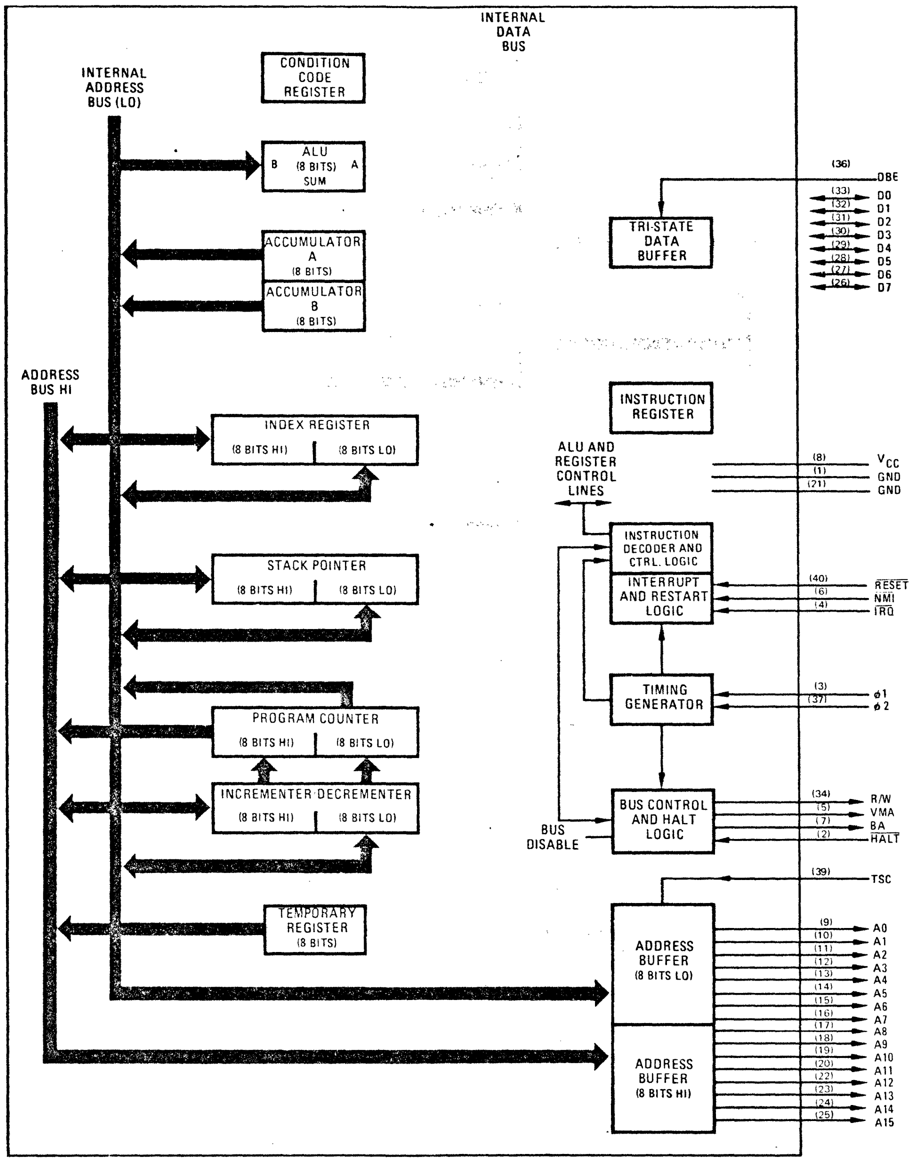


FIGURE 1. BLOCK DIAGRAM OF S6800 MICROPROCESSOR

- Six Internal Registers — Two Accumulators, Index Register, Program Counter, Stack Pointer and Condition Code Register
- Direct Memory Access (DMA) and Multiple Processor Capability
- Clock Rates as High as 1 MHz
- Simple Bus Interface Without TTL
- Halt and Single Instruction Execution Capability

S6800 MICROPROCESSOR ARCHITECTURE

The S6800 Microprocessor (MPU) is an 8-bit parallel processor. It contains an 8-bit arithmetic unit (ALU), two 8-bit accumulators, one condition code register, and three 16-bit address storage registers, all of which are available for program use (see Figure 1). In addition, there are the following non-accessible registers: a 16-bit address incrementer/decrementer, an 8-bit temporary register and an 8-bit instruction register. There is also an instruction decode ROM and cycle control logic, interrupt and restart logic, bus control and halt logic, and a timing generator.

MPU PROGRAM ACCESSIBLE REGISTERS

Accumulators A and B — Two separate 8-bit accumulators that are used to hold operands and results of operations in the ALU.

Index Register — A 16-bit register used for memory address storage in Indexed Addressing operations.

Program Counter — A 16-bit register that holds the current program instruction address. Once the initial program starting address is loaded into the program counter, it is incremented under control of the MPU hardware.

Stack Pointer — A 16-bit register used for storage of the next available location in an external push-down/pop-up stack.

Condition Code Register — An 8-bit register that stores certain results of operations in the ALU. These bits are used as testable conditions for the conditional branch instructions. In addition, one bit position stores the interrupt mask bit and the two high order bits are unused. See Figure 2.

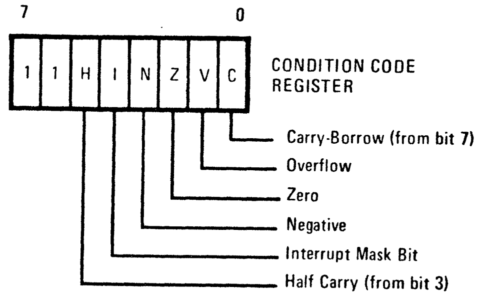


FIGURE 2.

EXTERNAL STACK MEMORY REGISTER — a push-down/pop-up stack that can be located anywhere in RAM and be of any convenient size. It is accessed with the stack pointer address and has several uses. First, it always stores the MPU register contents following an interrupt and return addresses during sub-routine execution. Second, it can also be used by the programmer to store data during program execution.

MPU HARDWARE REGISTERS (NOT ACCESSIBLE BY PROGRAM)

Instruction Register — 8-bit register used to receive and store all program instructions input into the MPU (via the data bus lines D0-D7).

Temporary Register — 8-bit register typically used to store the high order address bits prior to their output from the MPU onto the external address bus lines A8-A15.

Incrementer — 16-bit auxiliary address register, used by the MPU internal control logic, in conjunction with the program counter, to maintain and output the current program address.

MPU INTERNAL BUSSES

Within the MPU all data and address transfers between the registers, as well as to and from the ALU, are made across three internal 8-bit busses. The first is a data bus, the second is an address bus for the low order bits, and the third is an address bus for high order bits.

MPU INTERFACE DESCRIPTION

Signal	Pin	Function
Ø1	(3)	**Clocks Phase One and Phase Two** — Two pins are used for a two-phase non-overlapping clock that runs at the Vcc
Ø2	(37)	voltage level.
RESET	(4Ø)	**Reset** — This input is used to reset and start the MPU from a power down condition, resulting from a power failure or an initial start-up of the processor. If a positive edge is detected on the input, this will signal the MPU to begin the restart sequence. This will start execution of a routine to initialize the processor from its reset condition. All the higher order address lines will be forced high. For the restart, the last two (FFFE, FFFF) locations in memory will be used to load the program that is addressed by the program counter. During the restart routine, the interrupt mask bit is set and must be reset before the MPU can be interrupted by $\overline{IRQ}$. $\overline{Reset}$ must be held low for at least eight clock periods after VCC reaches 4.75 volts (Figure 4). If $\overline{Reset}$ goes high prior to the leading edge of Ø2, on the next Ø1 the first restart memory vector address (FFFE) will appear on the address lines. This

		location should contain the higher order eight bits to be stored into the program counter. Following, the next address FFFF should contain the lower order eight bits to be stored into the program counter.
VMA	(5)	**Valid Memory Address** — This output indicates to peripheral devices that there is a valid address on the address bus. In normal operation, this signal should be utilized for enabling peripheral interfaces such as the PIA and ACIA. This signal is not three-state. One standard TTL load and 3Ø pF may be directly driven by this active high signal.
AØ	(9)	**Address Bus** — Sixteen pins are used for the address bus. The outputs are three-state bus drivers capable of driving one standard TTL load and 13Ø pF. When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications.
A15	(25)	
TSC	(39)	**Three-State Control** — This input causes all of the address lines and the Read/Write line to go into the off or high impedance state. This state will occur 5ØØ ns after TSC = 2.4 V. The Valid Memory Address and Bus Available signals will be forced low. The data bus is not affected by TSC and has its own enable (Data Bus Enable). In DMA applications, the Three-State Control line should be brought high on the leading edge of the Phase One Clock. The Ø1 clock must be held in the high state and the Ø2 in the low state for this function to operate properly. The address bus will then be available for other devices to directly address memory. Since the MPU is a dynamic device, it can be held in this state for only 5.0 μs or destruction of data will occur in the MPU.
DØ	(33)	**Data Bus** — Eight pins are used for the data bus. It is bi-directional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers capable of driving one standard TTL load at 130 pF.
D7	(26)	
DBE	(36)	**Data Bus Enable** — This input is the three-state control signal for the MPU data bus and will enable the bus drivers when in the high state. This input is TTL compatible; however in normal operation, it can be driven by the phase two clock. During an MPU read cycle, the data bus drivers will be disabled internally. When it is desired that another device control the data bus such as in Direct Memory Access (DMA) applications, DBE should be held low.
R/W	(34)	**Read/Write** — This TTL compatible output signals the peripherals and memory devices whether the MPU is in a Read (high) or Write (low) state. The normal standby state of this signal is Read (high). Three-State Control going high will turn Read/Write to the off (high-impedance) state. Also, when the processor is halted, it will be in the off state. This output is capable of driving one standard TTL load and 130 pF.
$\overline{\text{HALT}}$	(2)	**Halt** — When this input is in the low state, all activity in the machine will be halted. This input is level sensitive. In the halt mode, the machine will stop at the end of an instruction, Bus Available will be at a one level, Valid Memory Address will be at a zero, and all other three-state lines will be in the three-state mode.
		Transition of the $\overline{\text{Halt}}$ line must not occur during the last 25Ø ns of phase one. To insure single instruction operation, the $\overline{\text{Halt}}$ line must go high for one Phase One Clock cycle.
BA	(7)	**Bus Available** — The Bus Available signal will normally be in the low state; when activated, it will go to the high state indicating that the microprocessor has stopped and that the address bus is available. This will occur if the $\overline{\text{Halt}}$ line is in the low state or the processor is in the WAIT state as a result of the execution of a WAIT instruction. At such time, all three-state output drivers will go to their off state and other outputs to their normally inactive level. The processor is removed from the WAIT state by the occurrence of a maskable (mask bit I = Ø) or nonmaskable interrupt. This output is capable of driving one standard TTL load and 3Ø pF.
$\overline{\text{IRQ}}$	(4)	**Interrupt Request** — This level sensitive input requests that an interrupt sequence be generated within the machine. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. At that time, if the interrupt mask bit in the Condition Code Register is not set, the machine will begin an interrupt sequence. The Index Register, Program Counter, Accumulators, and Condition Code Register are stored away on the stack. Next the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded

that points to a vectoring address which is located in memory locations FFF8 and FFF9. An address loaded at these locations causes the MPU to branch to an interrupt routine in memory.

The $\overline{\text{Halt}}$ line must be in the high state for interrupts to be recognized.

The $\overline{\text{IRQ}}$ has a high impedance pullup device internal to the chip; however a 3 kΩexternal resistor to Vcc should be used for wire-OR and optimum control of interrupts.

$\overline{\text{NMI}}$ (6) **Non-Maskable Interrupt** — A low-going edge on this input requests that a non-mask interrupt sequence be generated within the processor. As with the Interrupt Request signal, the processor will complete the current instruction that is being executed before it recognizes the $\overline{\text{NMI}}$ signal. The interrupt mask bit in the Condition Code Register has no effect on $\overline{\text{NMI}}$.

The Index Register, Program Counter, Accumulators, and Condition Code Register are stored away in the stack. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations FFFC and FFFD. An address loaded at these locations causes the MPU to branch to a non-maskable interrupt routine in memory.

$\overline{\text{NMI}}$ has a high impedance pullup resistor internal to the chip; however a 3 kΩexternal resistor to Vcc should be used for wire-OR and optimum control of interrupts.

Inputs $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ are hardware interrupt lines that are acknowledged during Ø2 and will start the interrupt routine on the Ø1 following the completion of an instruction.

MPU EXTERNAL BUSSES

The MPU communicates with its external memory and all I/O devices across an 8-bit bidirectional data bus, D0 through D7, and 16 address lines, A0 through A15. The MPU can be disconnected from either bus by two control signals DBE and TSC. In addition, a control bus maintains control of the bi directional Data Bus and provides access for control signals between the MPU and all external logic.

The MPU I/O bus relegates control to the programmed I/O devices, provides memory mapped I/O addressing and uses memory and register instructions to control all I/O operations. The MPU bus configuration is shown in Figure 3.

Programmed I/O Devices — The MPU relegates most of the I/O control to such I/O interfaces as the PIA or ACIA. Each of these circuits is programmable and can interface with peripheral devices without directly involving the MPU. For example, the MPU can preprogram a PIA to either output data to the MPU or to receive it. Thereafter, the PIA circuits assume all functions of interfacing with the peripherals and the MPU never has to look at the interface until service is required. It must service interrupts from the PIA, but never needs to wait for input data to become available or for output data to be accepted.

MEMORY MAPPED I/O Addressing — The I/O interfaces and memory are both located in the same address space within the S6800 system. The MPU can access any I/O device the same as a memory location

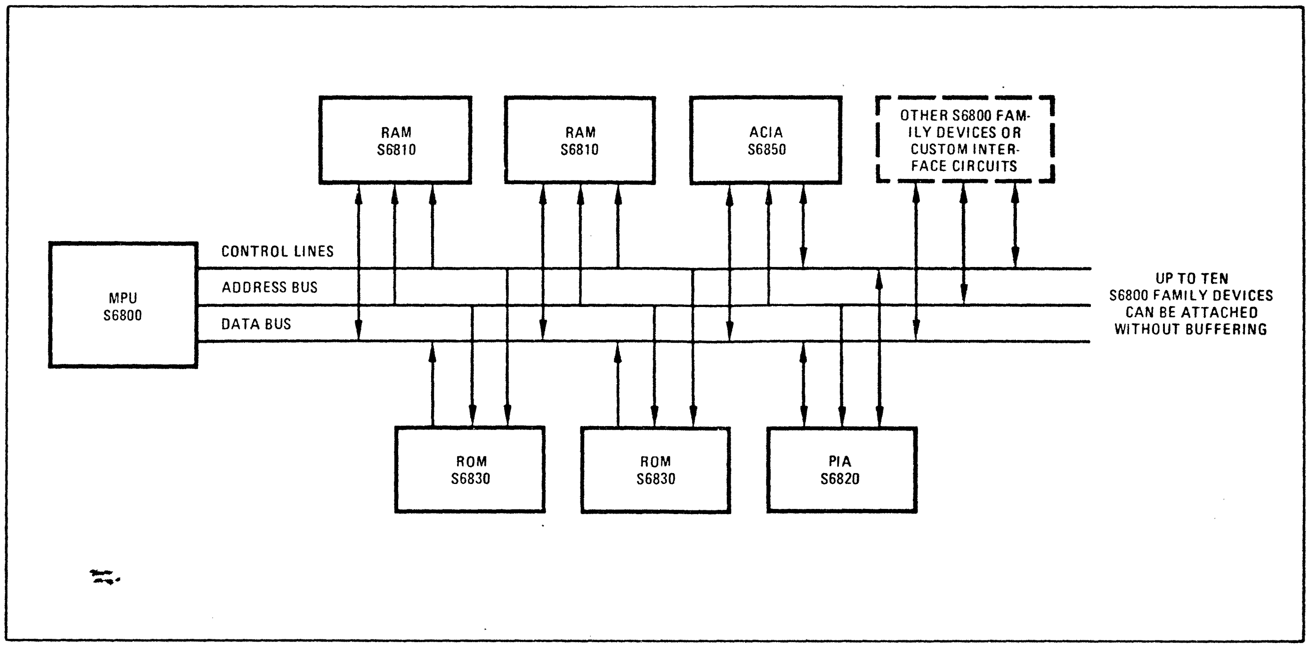

RAM S6810
RAM S6810
ACIA S6850
OTHER S6800 FAMILY DEVICES OR CUSTOM INTERFACE CIRCUITS
MPU S6800
CONTROL LINES
ADDRESS BUS
DATA BUS
UP TO TEN S6800 FAMILY DEVICES CAN BE ATTACHED WITHOUT BUFFERING
ROM S6830
ROM S6830
PIA S6820

FIGURE 3. S6800 BUS SYSTEM

— with address lines, instead of separate I/O control lines. Therefore, it can manipulate data in the I/O interface registers with the same programmed instructions as it uses for memory locations. This adds flexibility and increases system efficiency.

No Special I/O Instructions — The S6800 Instruction Set complements the above I/O addressing capability with specific instructions that can be used to access memory as well as I/O circuit registers and perform directly various manipulations on the data.

BUS INTERFACE — The bus interface consists of the Data Bus (DO-D7), the Address Bus (A0-A15), Read/Write (R/W), and Valid Memory Address (VMA). There are other signals which further control the operation of the MPU and thus affect the bus without actually being properly part of the bus interface; these include Data Bus Enable (DBE), Three-State Control (TSC), Halt and the interrupt control signals IRQ and NMI.

DATA BUS — The Data Bus comprises eight bidirectional data lines, which connect the MPU, all of the memory, and any I/O devices which may be addressed by the MPU. The MPU normally controls this bus during Ø2 time for the transfer of instructions and data into the MPU and the transfer of data out of the MPU. The direction of the data on the bus is a function of the R/W line generated by the MPU; a high on R/W constitutes a read, and the MPU accepts data during the latter portion of Ø2; a low on the R/W line is defined as a write out of the MPU, and the MPU drives the bus with the write data shortly after the low-to-high transition of DBE. Control of the Data Bus may be preempted from the MPU for Direct Memory Access (DMA) operations during Ø2 by operating the Halt line (Bus Available will go high when the bus is available for other than MPU-controlled use), or during Ø1 while DBE is low, and the MPU is not concerned with the contents of the data bus. When the MPU is not driving the data bus in a write operation, these lines are placed in a high impedance state to minimize the interference with other devices driving the bus; similarly when a memory or I/O device is not driving the bus in a read operation for which that device is selected, the bus lines in that device are placed in a high impedance state. At any one time only one device should be driving the bus, with all other connected system components in the high impedance state.

Address Bus — The Address bus comprises 16 address lines, by which the MPU identifies which of the 65,536 possible memory locations is to be read out or written into. In normal operation the MPU sets an address on the bus during Ø1 while TSC is low; this remains stable throughout Ø2 for the memory access operation. For DMA and other circumstances in which it is desired to control the Address Bus apart from the MPU, Ø1 may be extended during which time TSC may be set high; the MPU will respond by taking the Address Bus and R/W outputs to the high impedance state and outputting a low on VMA. A high on TSC also forces BA low.

The Valid Memory Address (VMA) output from the MPU should always be used in conjunction with the address on the Address Bus to determine whether the MPU is actually accessing memory (or peripheral registers), since in some circumstances the MPU will issue a temporary address in a read or write cycle which might be interpreted as a "false read" or a "false write" to some memory location. VMA may be thought of as a 17th bit of address, where only half of the addressable memory (i.e. the VMA bit = 1) is usable, although occasionally the MPU will attempt to read or write some location in the other half (i.e. VMA = 0). Thus it can be combined with the higher order address bits to select or deselect memory and peripherals, as required by the MPU at that time.

Bus Control Signals — If the VMA signal is not used to deselect memory and I/O registers (PIAs and ACIAs), false reads or false writes may result in ambiguous operation. These are of particular concern in the case of RAM memories and the I/O devices register (PIAs and ACIAs), since in the case of the ROM any false reads are ignored by the MPU and have no other effect, and false writes have no effect on the ROM. In the case of the RAM a false read also is of no concern, but if TSC or Halt is used or the MPU executes a WAI instruction the R/W line floats in the high impedance state which could be interpreted as a write by the RAM; the TST instruction actually results in a false write, but the data and address are the same as an immediately previous read, so the contents of RAM are not thus altered. For PIAs and ACIAs the VMA signal must be used in the selection logic, since a read from a PIA or ACIA register is used to clear an interrupt condition, and the failure to disable false reads could result in a missing interrupt. In the case of the PIA, VMA should not be used in the form of VMA-Ø2 for the Enable (E) input, since at least one E pulse is required before each active transition of the CA1 (or CB1, etc.) to detect the interrupt; a WAI instruction depending on this transition may thus lock out interrupts by setting VMA low. It is better to apply VMA to one of the Chip Select inputs to the PIA (CSO, CS1, or inverted to $\overline{CS2}$), or to specify the design to preclude the requirement that interrupts be detected on the trailing edge of a pulse.

Read/Write (R/W) is a control signal generated by the MPU to define the direction of the Data Bus. When low, the MPU is driving the Data Bus, and the selected memory or peripheral should accept the data written into it; when R/W is high, the selected memory or peripheral is being read into the MPU, and should be driving the bus. This control goes into the high impedance state when the Address Bus is disabled by TSC.

R/W is routed to the various memory and peripheral components as part of the system control. ROMs should be disabled when R/W is low, since they cannot be written into. RAMs and PIAs use the R/W signal to distinguish read and write operations. The ACIA has four internal registers, of which two are selected in the read operation, and two are selected for write operations, by the connection of R/W to the appropriate ACIA input.

MPU Operating Cycle

Instructions are executed within the MPU in incremental time periods (MPU cycles), each consisting of one Ø1 clock period and one Ø2 clock period. When the MPU is operating on a 1 MHz input clock, each MPU cycle is 1 microsecond long. It takes a minimum of two MPU cycles or 2 microseconds to execute a single word instruction.

During the Ø1 period the MPU typically outputs a memory address to access (fetch) one 8-bit program instruction or data byte and then, during Ø2, loads the byte into an internal register. During the next Ø1 period the MPU executes the associated internal operation with the ALU and the registers. With this fetch-execute sequence an instruction may be completed in only two MPU cycles, or may require as many as 12. While the MPU is executing successive cycles, it is also common for it to overlap functions. For example, during any given clock period the MPU may be executing one instruction in the ALU or registers; while at the same time a fetch is being performed with the address in the program counter.

Program Control

These internal operations of the MPU, as well as the output of address, data, and control signals, are all managed by the instruction decode and control logic. For example, to perform the execute part of any instruction, the control logic circuits generate signals that cause the ALU to perform addition, subtraction, or some Boolean logic function. These signals can also cause the contents of one register to be transferred into another, a register to be simply incremented or decremented, or some other similar function to occur. Such ALU and register operations are used to execute all of the S6800 instructions.

MPU Addressing Modes

The S6800 eight-bit microprocessing unit has seven address modes that can be used by a programmer, with the addressing mode a function of both the type of instruction and the coding within the instruction. During the fetch part of any MPU cycle a memory address is required in order to access a particular location in the external memory. This address is normally stored in the program counter. The program counter is 16 bits wide and therefore, can address any one of a maximum of 65,536 bytes.

At the beginning of a program sequence the MPU is initialized and the beginning address is loaded into the program counter. From there on the program counter is incremented automatically, so that at the end of any instruction cycle it stores the next instruction address. If the program contains an instruction to branch or jump to a different memory location, the op code must be followed by two bytes which load the new address into the program counter. There are, however, addressing techniques with which the jump can be accomplished by fetching only one new address byte out of the memory. For example, if the destination of a branch is within 129 locations forward of 125 locations back of the current program counter contents, Relative Addressing can be used. In this mode only the op code and one signed 8-bit byte is fetched from the memory and is added to the program counter contents.

In Indexed Addressing, a single byte is added to the contents of the index register and the result is transferred into the program counter. Thus, the above addressing variations can be used to reduce the number of bytes that need be fetched to generate a new address. This reduces the number of MPU cycles and speeds up program execution.

The various addressing modes can also be used in a similar manner to generate the source or destination addresses for data. MPU addressing modes are summarized in the following;

INSTRUCTION FORMAT

BYTE # 1	ONE WORD INSTRUCTION

BYTE # 1	BYTE # 2	TWO WORD INSTRUCTIONS

BYTE # 1	BYTE # 2	BYTE # 3	THREE WORD INSTRUCTIONS

ACCUMULATOR ADDRESSING (ACCX)

OP CODE

A single byte instruction addressing operands only in accumulator A or accumulator B.

IMPLIED ADDRESSING

OP CODE

Single byte instruction where the operand address is implied by the instruction definition (i.e., Stack Pointer, Index Register or Condition Register).

IMMEDIATE ADDRESSING

OP CODE	IMMEDIATE OPERAND

HIGHER	IMMEDIATE OPERAND	IMMEDIATE OPERAND LOWER

Two or three byte instructions with an eight or sixteen bit operand respectively. For accumulator operations the eight bit operand is contained in the second byte of a two byte instruction. For Index Register operations (e.g. LDX) sixteen bit operand is contained in the second and third byte of a three byte instruction.

DIRECT ADDRESSING

OP CODE	ADDRESS 0-255

Two byte instructions with the address of the operand contained in the second byte of the instruction. This format allows direct addressing of operands within the first 256 memory locations.

EXTENDED ADDRESSING

OP CODE	ADDRESS HIGHER	ADDRESS LOWER

Three byte instructions with the higher eight bits of the operand address contained in the second byte and the lower eight bits of address contained in the third byte of the instruction. This format allows direct addressing of all 65,536 memory locations.

INDEXED ADDRESSING

OP CODE	INDEX ADDRESS

Two byte instructions where the 8 bit unsigned address contained in the second byte of the instruction is added to the sixteen bit Index Register resulting in a sixteen bit effective address. The effective address is stored in a temporary register and the contents of the Index Register are unchanged.

RELATIVE ADDRESSING

OP CODE	RELATIVE ADDRESS

Two byte instructions where the relative address contained in the second byte of the instruction is added to the sixteen bit program counter plus two. The relative address is interpreted as a two's complement number allowing relative addressing within a range of −125 to +129 bytes of the present instruction.

Interrupts.

The S6800 MPU can be interrupted by any of several signals and program instructions, each of which initiates a different sequence in the MPU. Including the Reset signal, there are four interrupts — three hardware interrupts (signal lines connected to the MPU) and one software interrupt (SWI instruction). Each class of interrupt is described in the following;

Nonmaskable Interrupt (NMI) — initiated by a low-going signal on the $\overline{\text{NMI}}$ line to the MPU; always interrupts the MPU — even while another interrupt is being processed and the interrupt mask bit is set. Therefore, NMI can be considered to the highest priority interrupt. It causes the following sequence of events:

1. At the completion of the instruction being executed, the contents of the program accessible registers (Figure 3) are stored in the stack.
2. The interrupt mask bit is set.
3. Starting with its next cycle, the MPU accesses locations FFFC and FFFD in the memory and loads the contents into the program counter.

Interrupt Request (IRQ) — initiated by a logic low signal on the $\overline{IRQ}$ line; interrupts the MPU as long as the interrupt mask bit is not set. It causes the following sequence of events:

1. At the completion of the instruction being executed, the interrupt mask bit is tested. If the bit is set the interrupt must wait; if it is not set, contents of the program accessible registers are stored in the stack.
2. The interrupt mask bit is set.
3. Starting with the next cycle, the MPU accesses locations FFFA and FFFB in the memory and loads the contents into the program counter.

Software Interrupt (SWI) — initiated by the SWI instruction and causes the following sequence of events:

1. Contents of the program accessible registers are stored in the stack.
2. The interrupt mask bit is set.
3. Starting with the next cycle, the MPU accesses locations FFFE and FFFF in the memory and loads the contents into the program counter.

Reset — initiated by a positive going edge on the RESET line to the MPU. It causes the following sequence of events:

1. All program accessible registers are cleared and other circuits in the MPU are initialized.
2. The interrupt mask bit is set.
3. Starting with the next cycle, the MPU accesses locations FFFE and FFFF in the memory and loads the contents into the program counter.

Wait (WAI) — an instruction that causes the MPU to stop all processing and wait for a hardware interrupt. This instruction is not an interrupt in itself because it does not cause branching to any memory address, however, it does cause contents of the program accessible registers to be stored into the stack, in preparation for an interrupt.

All interrupts are vectored — they cause the MPU to automatically access a predetermined location in the memory and fetch a branch address of the routine or program to which the MPU is to go to service the interrupt. All interrupts except Reset also cause the contents of each program accessible MPU register (with the exception of the stack pointer) to be transferred to the external stack and thus be saved for later processing. The IRQ interrupt is also maskable — it cannot interrupt the MPU as long as bit 4 in the condition code register is set.

The S6800 requires a 16-bit vector address to indicate the location of routines for Restart, Non-maskable Interrupt, and Maskable Interrupt. Additionally an address is required for the Software Interrupt Instruction (SWI). The processor assumes the uppermost eight memory locations, FFF8 — FFFF, are assigned as interrupt vector addresses as defined in Figure 4.

FIGURE 4. MEMORY MAP FOR INTERRUPT VECTORS

Vector		Description
MS	LS	
FFFE	FFFF	Restart
FFFC	FFFD	Non-maskable Interrupt
FFFA	FFFB	Software Interrupt
FFF8	FFF9	Interrupt Request

After completing the current instruction execution the processor checks for an allowable interrupt request via the $\overline{IRQ}$ or $\overline{NMI}$ inputs as shown by the simplified flow chart in Figure 5.

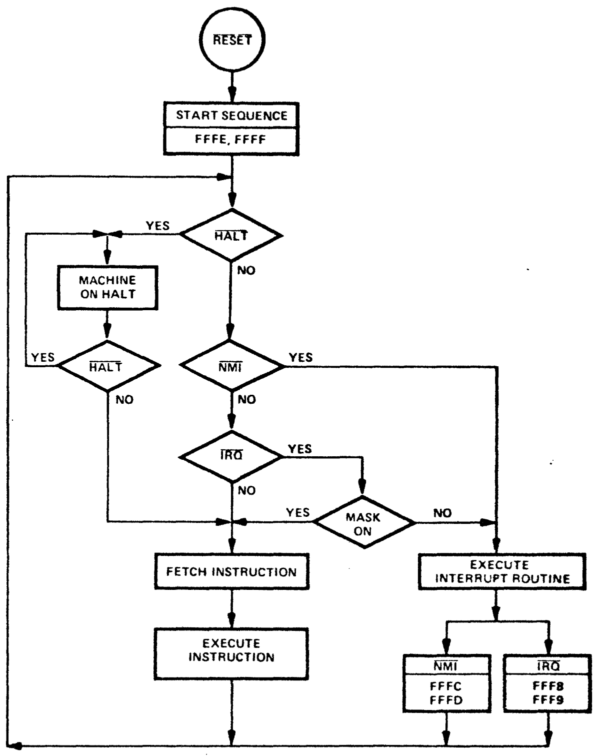
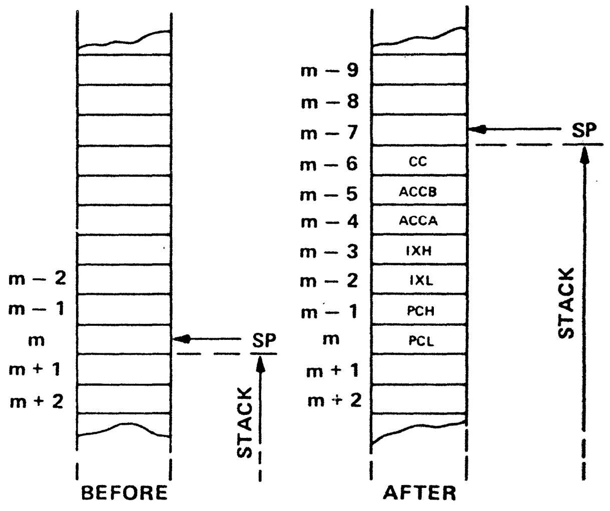


FIGURE 5. MPU FLOW CHART



SP =	Stack Pointer
CC =	Condition Codes (Also called the Processor Status Byte)
ACCB =	Accumulator B
ACCA =	Accumulator A
IXH =	Index Register, Higher Order 8 Bits
IXL =	Index Register, Lower Order 8 Bits
PCH =	Program Counter, Higher Order 8 Bits
PCL =	Program Counter, Lower Order 8 Bits

FIGURE 6. SAVING THE STATUS OF THE MICROPROCESSOR IN THE STACK

Recognition of either external interrupt request or a Wait for Interrupt (WAI) or Software Interrupt (SWI) instruction causes the contents of the Index Register, Program Counter, Accumulators and Condition Code Register to be transferred to the stack as shown in Figure 6.

S6800 INSTRUCTION SET

The S6800 MPU has a set of 72 basic instructions, listed in alphabetical order in Table 1. These include binary and decimal arithmetic functions, as well as logical, shift, rotate, load, store, branch, interrupt, and stack manipulation functions. Most of the instructions have several variations and most can be used with several memory addressing modes. Thus, the total complex of instructions available to the programmer actually is 197.

An instruction can be from one to three bytes long, depending on the addressing mode used with the instruction. The first byte always contains the operation code, which designates the kind of operation the MPU will perform. In single byte instructions no memory address is required, because the operation is performed on one of the internal MPU registers. In multiple byte instructions the second and third byte can be the operand, or a memory address for the operand.

A noteworthy feature of the S6800 MPU is that some of the instructions can operate directly on any memory location. In other computer systems it is common that the processor fetches an operand from memory, stores it in the accumulator, then executes

ABA	Add Accumulators	**INS**	Increment Stack Pointer
ADC	Add with Carry	**INX**	Increment Index Register
ADD	Add	**JMP**	Jump
AND	Logical And	**JSR**	Jump to Subroutine
ASL	Arithmetic Shift Left	**LDA**	Load Accumulator
ASR	Arithmetic Shift Right	**LDS**	Load Stack Pointer
BCC	Branch if Carry Clear	**LDX**	Load Index Register
BCS	Branch if Carry Set	**LSR**	Logical Shift Right
BEQ	Branch if Equal to Zero	**NEG**	Nagate
BGE	Branch if Greater or Equal Zero	**NOP**	No Operation
BGT	Branch if Greater than Zero	**ORA**	Inclusive OR Accumulator
BHI	Branch if Higher	**PSH**	Push Data
BIT	Bit Test	**PUL**	Pull Data
BLE	Branch if Less or Equal	**ROL**	Rotate Left
BLS	Branch if Lower or Same	**ROR**	Rotate Right
BLT	Branch if Less than Zero	**RTI**	Return from Interrupt
BMI	Branch if Minus	**RTS**	Return from Subroutine
BNE	Branch if Not Equal to Zero	**SBA**	Subtract Accumulators
BPL	Branch if Plus	**SBC**	Subtract with Carry
BRA	Branch Always	**SEC**	Set Carry
BSR	Branch to Subroutine	**SEI**	Set Interrupt Mask
BVC	Branch if Overflow Clear	**SEV**	Set Overflow
BVS	Branch if Overflow Set	**STA**	Store Accumulator
CBA	Compare Accumulators	**STS**	Store Stack Register
CLC	Clear Carry	**STX**	Store Index Register
CLI	Clear Interrupt Mask	**SUB**	Subtract
CLR	Clear	**SWI**	Software Interrupt
CLV	Clear Overflow	**TAB**	Transfer Accumulators
CMP	Compare	**TAP**	Transfer Accumulators to Condition Code Reg.
COM	Complement	**TBA**	Transfer Accumulators
CPX	Compare Index Register	**TPA**	Transfer Condition Code Reg. to Accumulator
DAA	Decimal Adjust	**TST**	Test
DEC	Decrement	**TSX**	Transfer Stack Pointer to Index Register
DES	Decrement Stack Pointer	**TXS**	Transfer Index Register to Stack Pointer
DEX	Decrement Index Register	**WAI**	Wait for Interrupt
FOR	Exclusive OR		
INC	Increment		

TABLE 1. S6800 MICROPROCESSOR INSTRUCTION SET

the operation in the ALU, and finally writes the result back into the memory. The S6800 is able to accomplish the same with only a single instruction, because it operates with any external location in the same manner as with an internal register. For example, it can directly increment or decrement the contents of a memory location. Because the MPU addresses I/O devices just like a memory location, it can do the same with registers inside the PIA or ACIA. The ASL, ASR, LSR, and ROL are other examples of instructions which operate in this manner.

S6810 — 128 X 8 STATIC READ/WRITE MEMORY

FUNCTIONAL DESCRIPTION

The S6810 is a static 128 X 8 Read/Write Memory designed and organized to be compatible with the S6800 Microprocessor. Interfacing to the S6810 consists of an 8 Bit Bidirectional Data Bus, Seven Address Lines, a single Read/Write Control line, and six Chip Enable lines, four negative and two positive.

For ease of use, the S6810 is a totally static memory requiring no clocks or cell refresh. The S6810 is fabricated with N channel silicon gate technology to be fully DTL/TTL compatible with only a single +5 volt power supply required.See Figure 7 for Funtional Block Diagram.

S6820 — PERIPHERAL INTERFACE ADAPTER (PIA)

FUNCTIONAL DESCRIPTION

The S6820 Peripheral Interface Adapter provides the universal means of interfacing peripheral equipment to the S6800 Microprocessing Unit (MPU). This device is capable of interfacing the MPU to peripherals through two I/O 8-bit bidirectional peripheral data buses and four control lines. No external logic is required for interfacing to most peripheral devices.

The functional configuration of the PIA is programmed by the MPU during system initialization. Each of the peripheral data lines can be programmed to act as an input or output, and each of the four control/interrupt request lines may be programmed for one of several control modes. This allows a high degree of flexibility in the over-all operation of the interface.

The PIA interfaces to the S6800 MPU with an eight-bit bidirectional data bus, three chip select lines, two register select lines, two interrupt request lines, read/write line, enable line and reset line. These signals, in conjunction with the S6800 VMA output, permit the MPU to have complete control over the PIA. VMA may be utilized to gate the input signals to the PIA. See Figure 8 for Funtional Block Diagram.

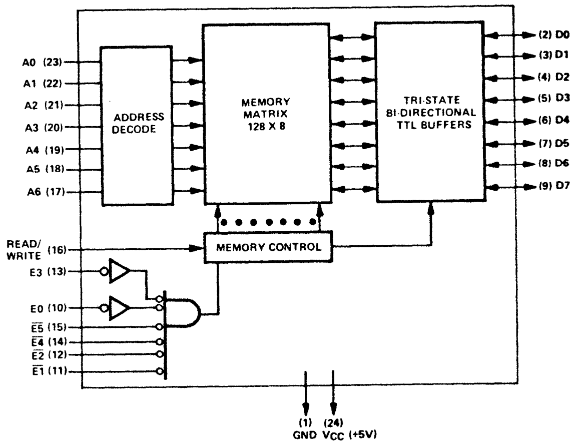


FIGURE 7. FUNCTIONAL BLOCK DIAGRAM

FEATURES

- Organized as 128 Bytes of 8 Bits
- Static Operation
- Bi-Directional Three-State Data Input/Output
- Six Chip Enable Inputs (Four Active Low, Two Active High)
- Single 5-Volt Power Supply
- TTL Compatible
- Maximum Access Time = 1.0μs for S6810 575 ns for S6810-1

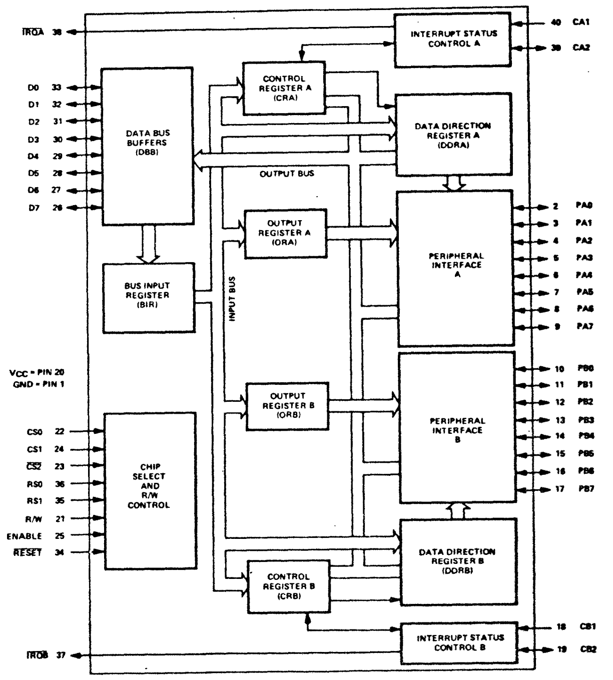


FIGURE 8. FUNCTIONAL BLOCK DIAGRAM

FEATURES

- 8-Bit Bidirectional Data Bus for Communication with the MPU
- Two Bidirectional 8-Bit Buses for Interface to Peripherals
- Two Programmable Control Registers
- Two Programmable Data Direction Registers
- Four Individually-Controlled Interrupt Input Lines; Two Usable as Peripheral Control Outputs
- Handshake Control Logic for Input and Output Peripheral Operation
- High-Impedance 3-State and Direct Transistor Drive Peripheral Lines
- Program Controlled Interrupt and Interrupt Disable Capability

S6830 — 1024 X 8 READ ONLY MEMORY

FUNCTIONAL DESCRIPTION

The S6830 is a mask programmable read only memory organized 1024 words x 8 bits for application in byte organized systems. The S6830 is totally bus compatible with the S6800 microprocessor. Interfacing to the S6830 consists of an 8 bit three-state data bus, four mask programmable chip selects and ten address lines.

The S6830 is a totally static memory requiring no clocks. Access time is compatible with maximum data rates in a S6800 microprocessor system. The device operates from a single +5 volt power supply and is fabricated with N channel silicon gate technology. See Figure 9 for Function Block Diagram.

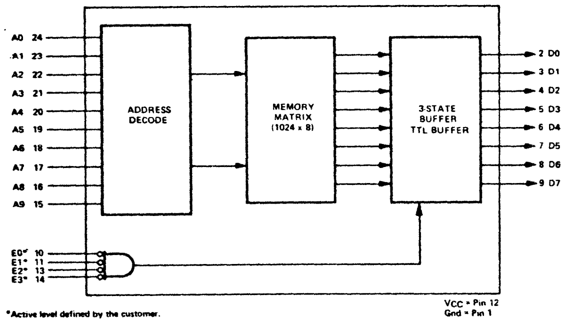


FIGURE 9. FUNCTIONAL BLOCK DIAGRAM

FEATURES

- Organized as 1024-Bytes of 8 Bits
- Static Operation
- Three-State Data Output
- Four Chip Enable Inputs (Mask Programmable)
- Single 5-Volt Power Supply
- TTL Compatible Input/Output
- Maximum Access Time = 575 ns

S6831/A/B/C — 2048 X 8 READ ONLY MEMORY

FUNCTIONAL DESCRIPTION

The S6831/A/B/C is a 16,384 bit mask programmable MOS Read Only Memory organized 2K words x 8 bits. This ROM has been designed to supply large bit storage, high performance memory for microprocessors and other demanding applications with simple interface requirements. The device will operate from a single +5V supply and is manufactured with a N-channel silicon gate depletion load technology. This device is available in all common high density ROM pinouts. See Figure 10 for Functional Block Diagram.

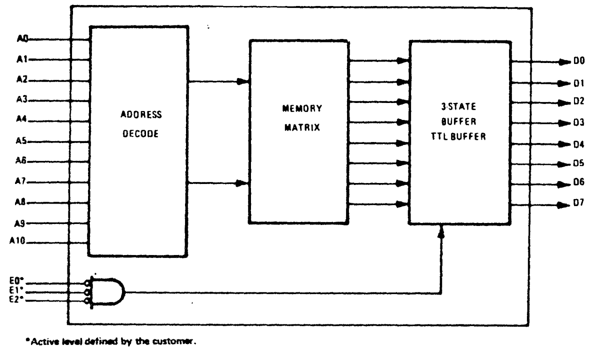


FIGURE 10. FUNCTIONAL BLOCK DIAGRAM

FEATURES

- Mask programmable
- Maximum Access Time = 450 ns@CL = 130 pF
- Low Power 150 mW avg.
- Organized as 2048-Bytes of 8 Bits
- Static Operation
- Three-State Data Output
- 3 Chip Enable Inputs (Mask Programmable)
- The S6831 is pinout similar with the S6830
- The S6831A is pinout compatible with the 2316A, 8316A
- The S6831B is pinout compatible with the Intel 2316B, MC68317
- The S6831C is pinout compatible with the EA4600
- Single 5-Volt Power Supply
- TTL Compatible Input/Output

S6834 — 512 X 8 BIT EPROM

FUNCTIONAL DESCRIPTION

The S6834 is a high speed, static, 512 x 8 bit, erasable and electrically programmable read only memory designed for the in bus-organized systems. Both input and output are TTL compatable during both read and write modes. Packaged in a 24 pin hermetically sealed dual in-line package the bit pattern can be erased by exposing the chip to an ultra-violet light source through the transparent lid, after which a new pattern can be written. See Figure 11 for Funtional Block Diagram.

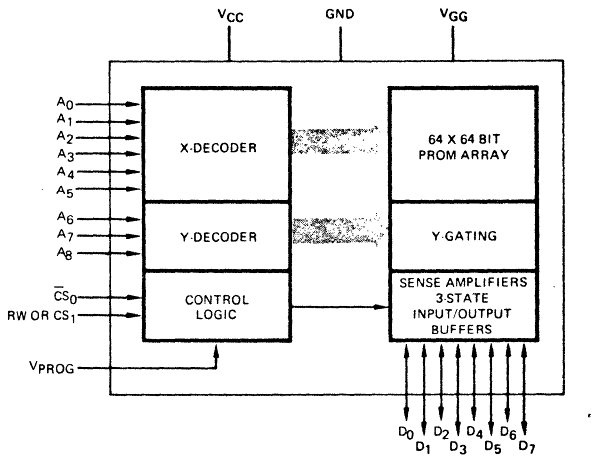


FIGURE 11. FUNCTIONAL BLOCK DIAGRAM

FEATURES

- On-Board Programmability
- Fast Access Time — 575 ms Max.
- Pin Configuration Similar to the S6830 1K x 8 Bit ROM
- High Speed Programming — Less than 1 Minute for all 4096 Bits
- Programmed with R/W, CS and VPROG Pins
- Completely TTL Compatible — Excluding the VPROG Pin
- Ultraviolet Light Erasable — Less than 10 Minutes
- Static Operation — No Clocks Required
- Three-State Data I/O
- Standard Power Supplies +5V and −12V
- Mature P-Chan Process

S6850 — ASYNCHRONOUS COMMUNICATION INTERFACE ADAPTER (ACIA)

FUNCTIONAL DESCRIPTION

The S6850 Asynchronous Communications Interface Adapter (ACIA) provides the data formatting and control to interface serial asynchronous data communications to bus organized systems such as the S6800 Microprocessing Unit.

The S6850 includes select enable, read/write, interrupt and bus interface logic to allow data transfer over an eight bit bi-directional data bus. The parallel data of the bus system is serially transmitted and received by the asynchronous data interface, with proper formatting and error checking. The functional configuration of the ACIA is programmed via the data bus during system initialization. Word lengths, clock division ratios and transmit control through the Request to Send output may be programmed. For modem operation three control lines are provided. These lines allow the ACIA to interface directly with the S6860 0-600 bps digital modem. See Figure 12 for Functional Block Diagram.

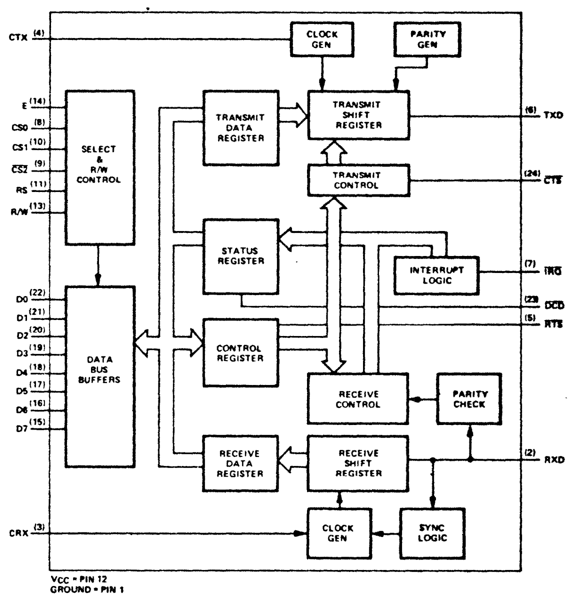


FIGURE 12. FUNCTIONAL BLOCK DIAGRAM

FEATURES

- Eight and nine-bit transmission with optional even and odd parity.
- Parity, overrun and framing error checking.

- Programmable control register.
- Optional÷1,÷16, and÷64 clock modes.
- Up to 500,000 bps transmission.
- 8 Bit Bidirectional Data Bus for Communication with MPU.
- False start bit deletion.
- Peripheral/modem control functions.
- Double buffered Receiver and Transmitter.
- One or two stop bit operation.

S2350 — UNIVERSAL SYNCHRONOUS RECEIVER/TRANSMITTER (USRT)

FUNCTIONAL DESCRIPTION

The S2350 Universal Synchronous Receiver Transmitter (USRT) is a single chip MOS/LSI device that totally replaces the serial to parallel and parallel to serial conversion logic required to interface a word parallel controller or data terminal to a bit-serial, synchronous communication network.

The USRT consists of separate receiver and transmitter sections with independent clocks, data lines and status. Common with the transmitter and receiver are word length and parity mode. Data is transmitted and received in a NRZ format at a rate equal to the respective input clock frequency.

Data messages are transmitted as a contiguous character stream, bit synchronous with respect to a clock and character synchronous with respect to framing or "sync" characters initializing each message. The USRT receiver compares the contents of the internal Receiver Sync Register with the incoming data stream in a bit transparent mode. When a compare is made, the receiver becomes character synchronous formatting a 5, 6, 7, or 8 bit character for output each character time. The receiver has an output buffer register allowing a full character time to transfer the data out. The receiver status outputs indicate received data available (RDA), receiver overrun (ROR), receive parity error (RPE) and sync character received (SCR). Status bits are available on individual output lines and can also be multiplexed onto the output data lines for bus organized systems. The data lines have tri-state outputs.

The USRT transmitter outputs 5, 6, 7, or 8 bit characters with correct parity at the transmitter serial output (TSO). The transmitter is buffered to allow a full character time to respond to a transmitter buffer empty (TBMT) request for data. Data is transmitted in a NRZ format changing on the positive transition of the transmitter clock (TCP). The character in the transmitter fill register is inserted into the data message if a data character is not loaded into the transmitter after a TBMT request. See Figure 13 for Functional Block Diagram.

FEATURES

- 500 KHz Data Rates
- Internal Sync Detection
- Fill Character Register
- Double Buffered Input/Output
- Bus Oriented Outputs
- 5-8 Bit Characters
- Odd/Even or No Parity

- Error Status Flags
- Single Power Supply (+5v)
- Input/Output TTL Compatible

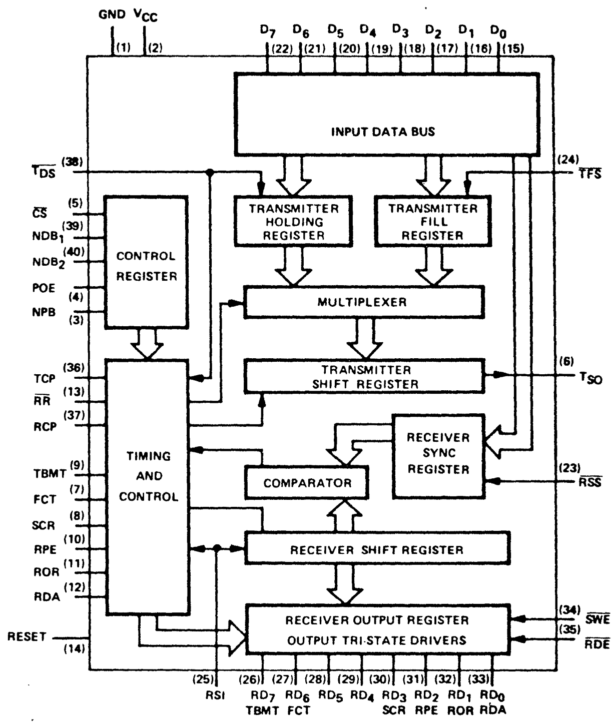


FIGURE 13. FUNCTIONAL BLOCK DIAGRAM

S6860 — 0-600 BPS DIGITAL MODEM

FUNCTIONAL DESCRIPTION

The S6860 is a 0-600 bps Digital Modem circuit designed to be integrated into a wide range of equipment utilizing serial data communications.

The modem provides the necessary modulation, demodulation and supervisory control functions to implement a serial data communications link, over a voice grade channel, utilizing frequency shift keying (FSK) a bit rates up to 600 bps. The S6860 can be implemented into a wide range of data handling systems, including stand alone modems, data storage devices, remote data communication terminals and I/O interfaces for minicomputers.

N-channel silicon gate technology permits the S6860 to operate using a single voltage supply and be fully TTL compatible.

The modem is compatible with the S6800 microcomputer family, interfacing directly with the Asynchronous Communications Interface Adapter (ACIA) to provide low-speed data communications capability. See Figure 14 for Functional Block Diagram.

FEATURES

- TTL compatible terminal interfaces
- Crystal/External reference control
- Compatible functions for 100 series data sets and 1001 A/B data couplers
- Full or half duplex operation
- Originate and answer mode
- Auto answer and disconnect
- Modem self test

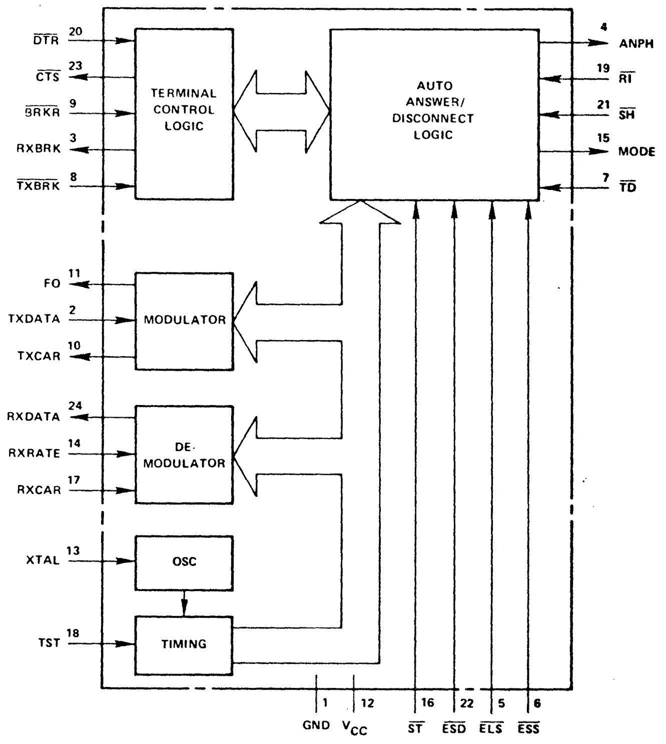


FIGURE 14. FUNCTIONAL BLOCK DIAGRAM

TYPICAL S6800 MICROCOMPUTER CONFIGURATION

The S6800 microcomputer functional IC components may be assembled in a modular building block manner into a very simple microcomputer system, or into any of progressively more complex systems, which can be used in many general or special purpose applications. The important feature of the S6800 family is that all microcomputer system components are directly compatible in signal functions, circuit performance characteristics, and logic levels. All operate on a single +5 Volt power supply.

A basic microcomputer system built with the S6800 functional components is shown in Figure 15. This basic microcomputer configuration includes a S6800 Microprocessor (MPU), IK bytes of ROM program storage, 128 bytes of RAM working storage and a two part input/output peripheral interface circuit.

Two-Phase Clock Circuitry and Timing — The MPU requires a two-phase non-overlapping clock which has a frequency range as high as 1 MHz. In addition to the two phases, this circuit should also generate an enable signal E, and its complement Ē, to enable ROMs, RAMs, PIAs and ACIAs. This Enable signal and its complement is obtained by ANDing Ø2 and VMA (Valid Memory Address).

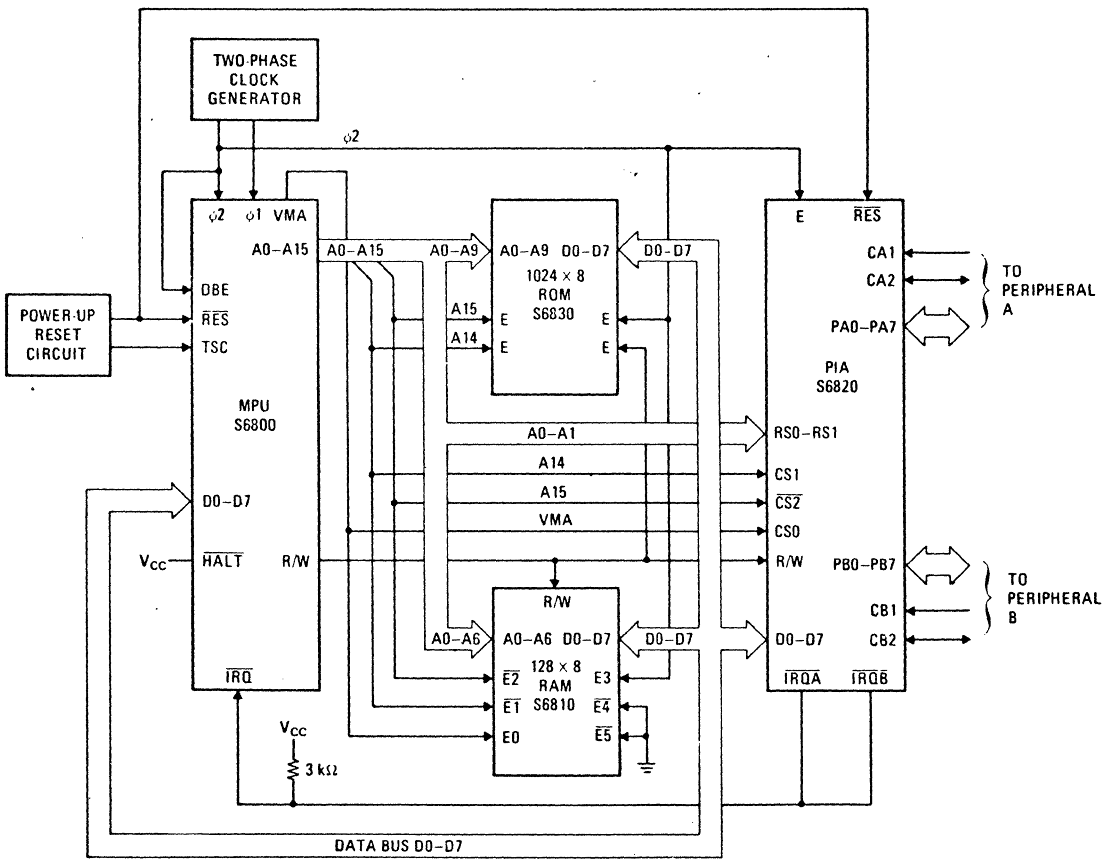


FIGURE 15. MINIMUM MICROCOMPUTER SYSTEM CONFIGURATION

Chip Selection and Addressing — The minimum system configuration permits direct selection of the ROM, RAM, ACIA and PIA without the use of special TTL select logic. This is accomplished by simply wiring the address lines A13 and A14 to the Enable or chip select lines on the memories and PIA. This permits the devices to be addressed as follows:

DEVICE	A14	A13	HEX ADDRESSES
RAM	0	0	0000–007F
PIA	0	1	2004–2007 (Registers)
ROM	1	1	6000–63FF

Other addressing schemes can be utilized which use any combination of two of the lines A10 through A14 for chip selection.

Peripheral Control — All control and timing for the peripherals that are connected to the PIA is accomplished by software routines under the control of the MPU.

Restart and Non-Maskable Interrupt — Since this basic system does not have a nonvolatile RAM, special circuitry to handle loss of power using NMI is not required. Circuitry is, however, required to insure proper initialization of the MPU when power is turned on. This circuit should insure that the Restart signal is held low for eight Ø1 clock cycles after the Vcc power supply reaches a voltage of approximately 4.75 volts DC. Also, in order to insure that a PIA or ACIA is not inadvertently selected during the power-on sequence, Three-State Control (TSC) should be held high until the positive transition of $\overline{\text{Restart}}$.

$\overline{\text{HALT}}$ — The $\overline{\text{Halt}}$ line is tied to Vcc and will automatically place the MPU in the run state when power is turned on. This signal may be used to halt the MPU if a switch is used to tie the line to ground for HALT and to Vcc for RUN.

The basic microcomputer system can be altered or expanded on in many different ways. For example, the S6850 Asynchronous Communication Interface Adapter (ACIA) can be substituted for a PIA, to enable the microcomputer to interface with a telecommunications modem. Or, additional memory can be added — either RAM or ROM — to expand the processing capability of the MPU. In general, the system can be expanded in a modular manner, by adding onto the bus as many as ten devices out of the S6800 family of modules. These additional modules can be any combination of memory or I/O IC circuits. In this manner a system of nearly any complexity and configuration can be assembled. Microcomputer system configurations requiring more than ten devices on the MPU bus require the addition of address and data bus buffers to operate at full speed.

By building your microcomputer from the S6800 family of devices, you take full advantage of the compatibility of the devices. They all conform to the MPU bus discipline, all are compatible in load levels, and the entire system runs on a common system clock. In effect you eliminate most all circuit design, save for the simple clock and power-up restart circuits. Because you are dealing with only a small number of integrated circuits, PCB circuit layout is simple and the entire microcomputer can be located on a single small circuit card.

ARTICLE BACKGROUND MATERIAL

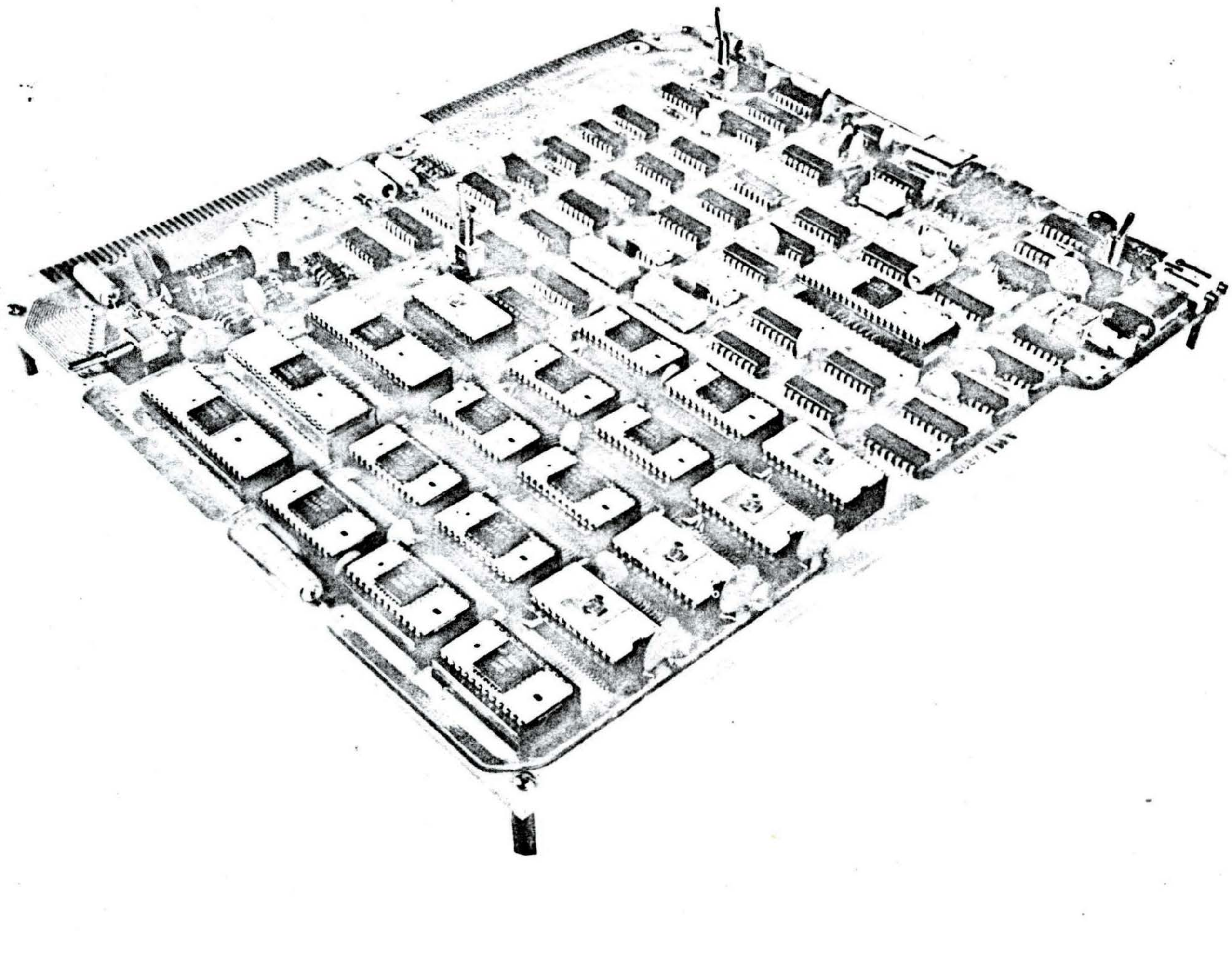
The majority of the material for this article was gleaned from AMI's excellent documentation with the intent of not redoing good work just for the sake of it. Hopefully I have organized this material and clarified it to the extent of making it clearer and easier to understand which was the intent.

Next month I will cover the hardware mechanization of the AMI, EVK Microcomputer Prototyping boards.

Want more information on AMI's microcomputer chip set? Write or call:

American Microsystems, Inc.
3800 Homestead Road
Santa Clara, Calif. 95051
Phone (408) 246-0330

AMI's EVK Series Microcomputer Prototyping Boards



By Robert A. Stevens

INTRODUCTION

This article is part #2 of a series of articles on the EVK Microcomputer hardware, firmware and supporting software. This month's article covers the EVK Microcomputer board architecture while last month's article described the functional architecture and characteristics of AMI's Microcomputer IC chip set.

EVK CONFIGURATIONS — The AMI EVK Microcomputer is a single board microcomputer mechanized with a standard S6800 MPU. The EVK Microcomputer comes in four basic configurations; EVK99, EVK100, EVK200, & EVK300, all of which use the same 10½" x 12" printed circuit board. EVK99 is a kit that includes the PCB and Microcomputer ICs consisting of one 6800 MPU, four 6810 RAM's, one 6820 PIA, two 6830 ROM, and one 6850 ACIA. EVK100 & EVK200 are kit configurations that include PCB, Microcomputer & T^{2}L IC's and differ from each other by the amount of hardware, memory and firmware (software in ROM) included with each configuration. EVK300 is the EVK200 kit with more EPROM memory and is factory assembled and tested. A Tiny BASIC Interpreter program is also available at no charge for the EVK300 Microcomputer board. Table 1, EVK Microcomputer Configuration Summary, shows the comparison between the different EVK configurations.

EVK BOARD CHARACTERISTICS	EVK 99	EVK 100	EVK200	EVK300
CPU	S6800	S6800	S6800	S6800
WORD SIZE	8 BITS	8 BITS	8 BITS	8 BITS
ADDRESS BUS	16 BITS (64K)	16 BITS (64K)	16 BITS (64K)	16 BITS (64K)
ROM	2K BYTES S6831 ROM	2K BYTES S6831 ROM	2K BYTES S6831 ROM	2K BYTES S6831 ROM
EPROM – VIRGIN	—	—	512 BYTES S6834 EPROM	2K BYTES S6834 EPROM
STATIC RAM	512 BYTES S6810 RAM	512 BYTES S6810 RAM	1K BYTES S6810 RAM	1K BYTES S6810 RAM
EPROM PROGRAMMING	—	—	PROGRAMS S6834 EPROM's	PROGRAMS S6834 EPROM's
I/O PORTS	1 PIA=2 PORTS – 8 BITS/PORT	—	3 PIA's=6 PORTS 8 BITS/PORT	3 PIA's=6 PORTS 8 BITS/PORT
ASR 33/35 TTY SERIAL INTERFACE	ACIA S6850	ACIA S6850 WITH 20ma CURRENT LOOP	ACIA S6850 WITH 20ma CURRNET LOOP	ACIA S6850 WITH 20 ma CURRENT LOOP
RS232C EIA SERIAL INTERFACE	ACIA S6850	ACIA S6800	ACIA S6850 WITH EIA RS232C	ACIA S6850 WITH EIA RS232C
INTERVAL TIMER (CRYSTAL)	—	—	1 ms & 100 μs TIME INTERVALS	1 ms & 100 μs TIME INTERVALS
MPU CRYSTAL CLOCK			INCLUDED	INCLUDED
CLOCK OUTPUTS (CRYSTAL)	—	16X BAUD RATE	2.4576 MHz, 1 MHz & 16X BAUD RATE	2.4576 MHz, 1 MHz & 16X BAUD RATE
DMA MODES	—	—	HALT MPU MODE, CYCLE STEAL MODE & MUX MODE	HALT MPU MODE, CYCLE STEAL MODE & MUX MODE
RESTART ADDRESS SELECTION	—	TWO 8 BIT DIP TOGGLE SWITCHES	TWO 8 BIT DIP TOGGLE SWITCHES	TWO 8 BIT DIP TOGGLE SWITCHES
TTY MONITOR SOFTWARE	PROTO ROM RESIDENT	PROTO ROM RESIDENT	PROTO ROM RESIDENT	PROTO ROM RESIDENT
SUBROUTINE PROGRAM LIBRARY SOFTWARE	RS^3 ROM RESIDENT	RS^3 ROM RESIDENT	RS^3 ROM RESIDENT	RS^3 ROM RESIDENT
ROM RESIDENT ASSEMBLER	\$35^{00} OPTION	\$35^{00} OPTION	\$35^{00} OPTION	\$35^{00} OPTION
OEM SINGLE QUANTITY PRICE	\$133^{00}	\$295^{00}	\$495^{00}	\$765^{00}

TABLE 1 EVK MICROCOMPUTER CONFIGURATION SUMMARY

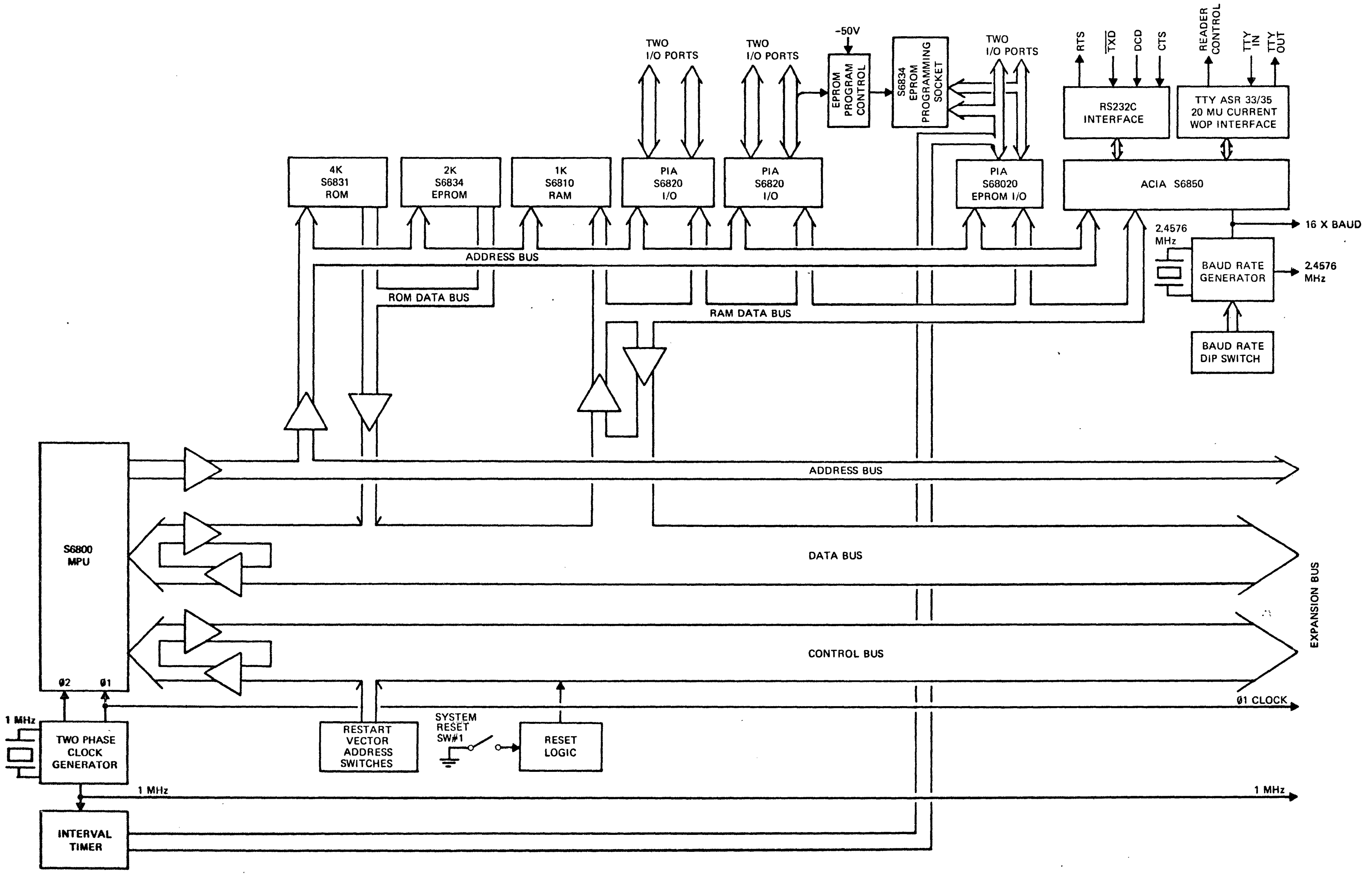
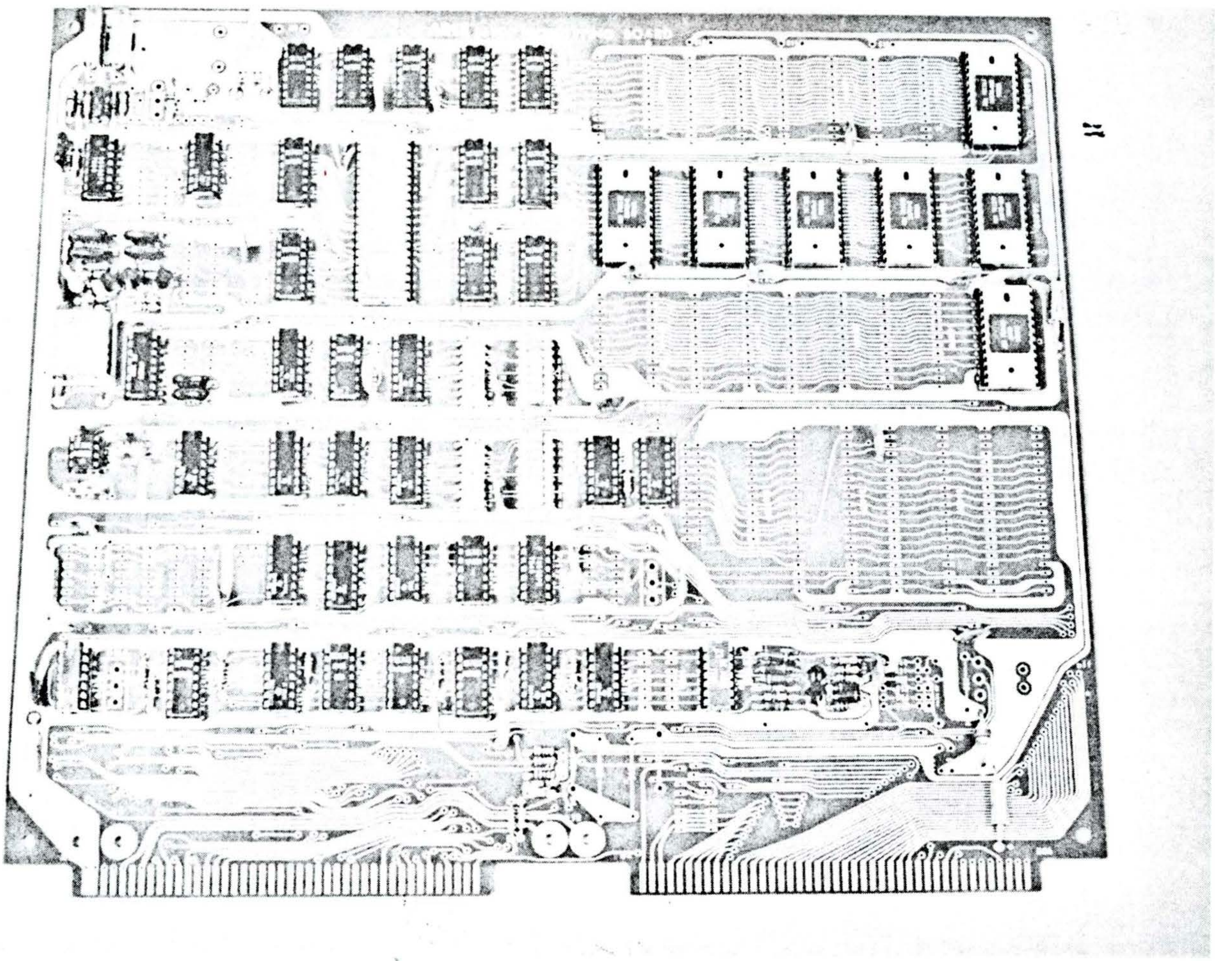


Figure 1. EVK Series Microcomputer Board



MAJOR EVK MICROCOMPUTER FEATURES

The common denominator EVK Microcomputer PCB provides the following on board major features when fully populated with hardware and software including options;

- 4K Bytes S6831 ROM memory (2K using S6831 ROM's)
- 2K Bytes S6834 EPROM memory
- 1 K Bytes S6810 Static RAM memory
- On board S6834 EPROM programming
- Six 8 bit PTA I/O ports
- 20 ma serial TTY current loop port interface
- RS232C EIA serial I/O port
- Switch selected baud rates to 19,200 bauds
- 1 MHz crystal or variable one shot MPU clock
- 5 crystal controlled timing signals available at PCB interface (2.4756 MHz 1MHz 16x baud rate, 100 μs & 1 ms) ms)
- Interrupt internal timing (100μs & 1 ms)
- Switch selectable MPU restart address
- 200 ms Power On Reset delay
- 3 DMA modes (HALT MPU, CYCLE STEAL & MUX)
- TTY PROTO Monitor System resident in ROM
- RS^3 ROM Subroutine Library resident in ROM
- ROM Resident Assembler — option
- Up to 40 ma @ 0.4V external bus loading
- 8T97 three state MPU bus drivers
- All MPU signal lines isolated & buffered
- System expansion via two 86 pin connectors

TYPICAL EVK MICROCOMPUTER APPLICATIONS

The EVK Microcomputer board allows the hardware development engineer, the logic designer, the programmer, the systems engineer, the mathematician, the scientist, the chemist or the hobbyist to have a complete working Microcomputer system, including development software by adding a low cost power supply and an ASR 33 TTY to the EVK Microcomputer board. The EVK series of Microcomputers boards allows the owner/user to use one of these boards to:

- Evaluate the complete set of AMI's family of Microcomputer IC's at a low investment of time & money — no design time is required.
- Serve as a general purpose Microcomputer for low volume systems to which the systems engineer can easily add additional I/O ports and memory.
- Serve as a low cost quick turn around prototype system to evaluate total system mechanization concept (hardware & software) and market acceptance prior to committing to a custom design system for large volume production.
- Serve as a low cost minimal 6800 Microcomputer application software development system.
- Serve as a low cost general purpose Microcomputer to run numerous application software programs.

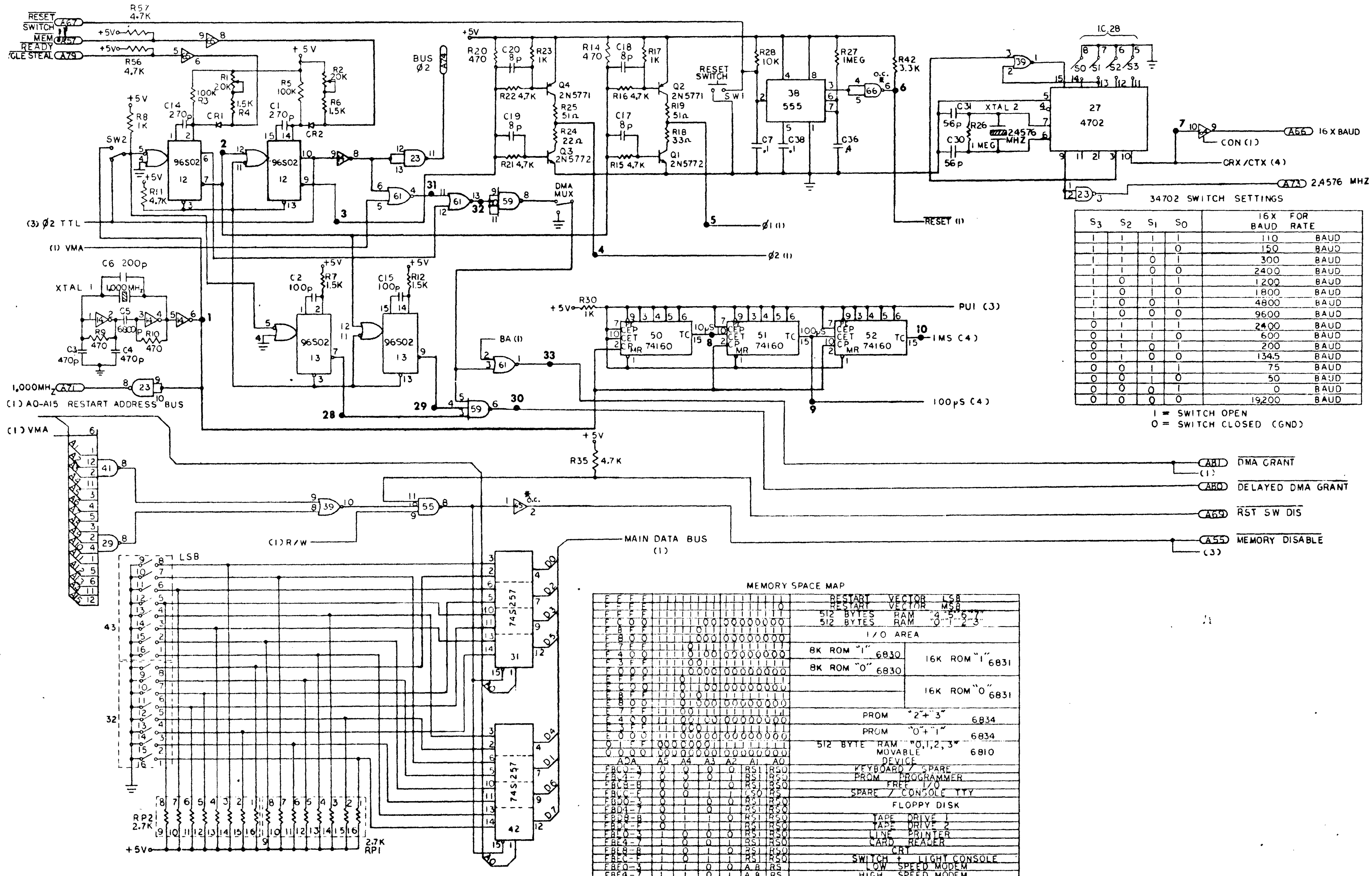


Figure 2. AMI S6800 Prototyping Board Schematic Diagram

34702 SWITCH SETTINGS

S_3	S_2	S_1	S_0	16X FOR BAUD RATE	
1	1	1	1	110	BAUD
1	1	1	0	150	BAUD
1	1	0	1	300	BAUD
1	1	0	0	2400	BAUD
1	0	1	1	1200	BAUD
1	0	1	0	1800	BAUD
1	0	0	1	4800	BAUD
1	0	0	0	9600	BAUD
0	1	1	1	2400	BAUD
0	1	1	0	600	BAUD
0	1	0	1	200	BAUD
0	1	0	0	134.5	BAUD
0	0	1	1	75	BAUD
0	0	1	0	50	BAUD
0	0	0	1	0	BAUD
0	0	0	0	19,200	BAUD

1 = SWITCH OPEN
0 = SWITCH CLOSED (GND)

MEMORY SPACE MAP

F F F F	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	RESTART VECTOR LSB	
F F F E	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 0	RESTART VECTOR MSB	
F F F F	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	512 BYTES RAM "4""5""6""7"	
F C 0 0	1 1 1 1 1 1 0 0	0 0 0 0 0 0 0 0	512 BYTES RAM "0""1""2""3"	
F 8 F F	1 1 1 1 1 0 1 1	1 1 1 1 1 1 1 1	I/O AREA	
F 8 0 0	1 1 1 1 1 0 0 0	0 0 0 0 0 0 0 0		
F 7 F F	1 1 1 1 0 1 1 1	1 1 1 1 1 1 1 1	8K ROM "1" 6830	16K ROM "1" 6831
F 4 0 0	1 1 1 1 0 1 0 0	0 0 0 0 0 0 0 0		
F 3 F F	1 1 1 1 0 0 1 1	1 1 1 1 1 1 1 1	8K ROM "0" 6830	
F 0 0 0	1 1 1 1 0 0 0 0	0 0 0 0 0 0 0 0		
E F F F	1 1 1 0 1 1 1 1	1 1 1 1 1 1 1 1		16K ROM "0" 6831
E C 0 0	1 1 1 0 1 1 0 0	0 0 0 0 0 0 0 0		
E 8 F F	1 1 1 0 1 0 1 1	1 1 1 1 1 1 1 1		
E 8 0 0	1 1 1 0 1 0 0 0	0 0 0 0 0 0 0 0		
E 7 F F	1 1 1 0 0 1 1 1	1 1 1 1 1 1 1 1	PROM "2"+"3" 6834	
E 4 0 0	1 1 1 0 0 1 0 0	0 0 0 0 0 0 0 0		
E 3 F F	1 1 1 0 0 0 1 1	1 1 1 1 1 1 1 1	PROM "0"+"1" 6834	
E 0 0 0	1 1 1 0 0 0 0 0	0 0 0 0 0 0 0 0		
0 1 F F	0 0 0 0 0 0 0 1	1 1 1 1 1 1 1 1	512 BYTE RAM "0,1,2,3" MOVABLE 6810	
0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0		

ADA	A5	A4	A3	A2	A1	A0	DEVICE
FBC0-3	0	0	0	0	RS1	RS0	KEYBOARD / SPARE
FBC4-7	0	0	0	1	RS1	RS0	PROM PROGRAMMER
FBC8-B	0	0	1	0	RS1	RS0	FREE I/O
FBCC-F	0	0	1	1	CS0	RS	SPARE / CONSOLE TTY
FBD0-3	0	1	0	0	RS1	RS0	FLOPPY DISK
FBD4-7	0	1	0	1	RS1	RS0	
FBD8-B	0	1	1	0	RS1	RS0	TAPE DRIVE 1
FBDC-F	0	1	1	1	RS1	RS0	TAPE DRIVE 2
FBE0-3	1	0	0	0	RS1	RS0	LINE PRINTER
FBE4-7	1	0	0	1	RS1	RS0	CARD READER
FBE8-B	1	0	1	0	RS1	RS0	CRT
FBEC-F	1	0	1	1	RS1	RS0	SWITCH + LIGHT CONSOLE
FBF0-3	1	1	0	0	A B	RS	LOW SPEED MODEM
FBF4-7	1	1	0	1	A B	RS	HIGH SPEED MODEM
FBF8-B	1	1	1	0	RS1	RS0	PAPER TAPE READER
FBFC-F	1	1	1	1	RS1	RS0	PAPER TAPE PUNCH

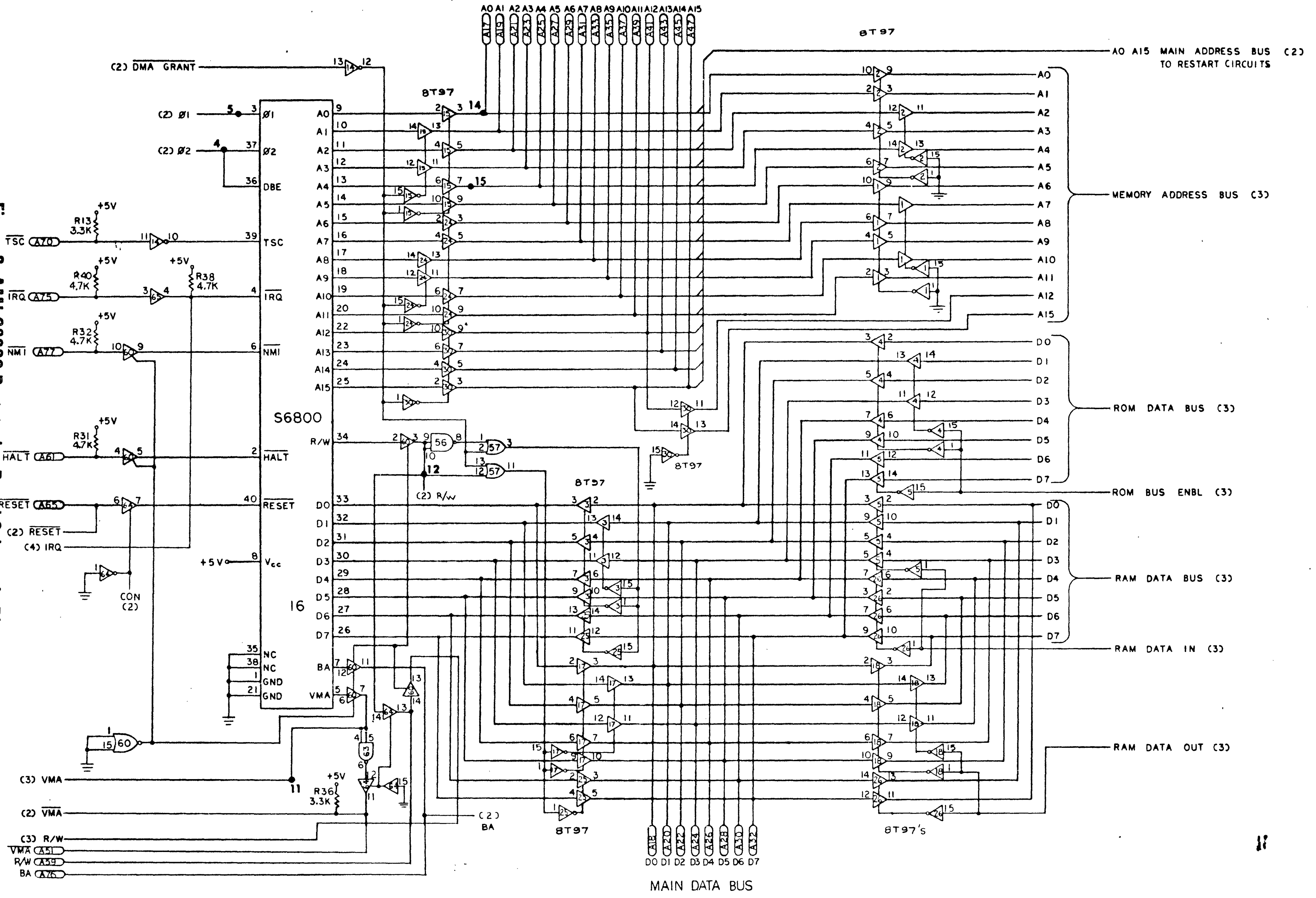


Figure 3. AMI S6800 Prototyping Board Schematic Diagram

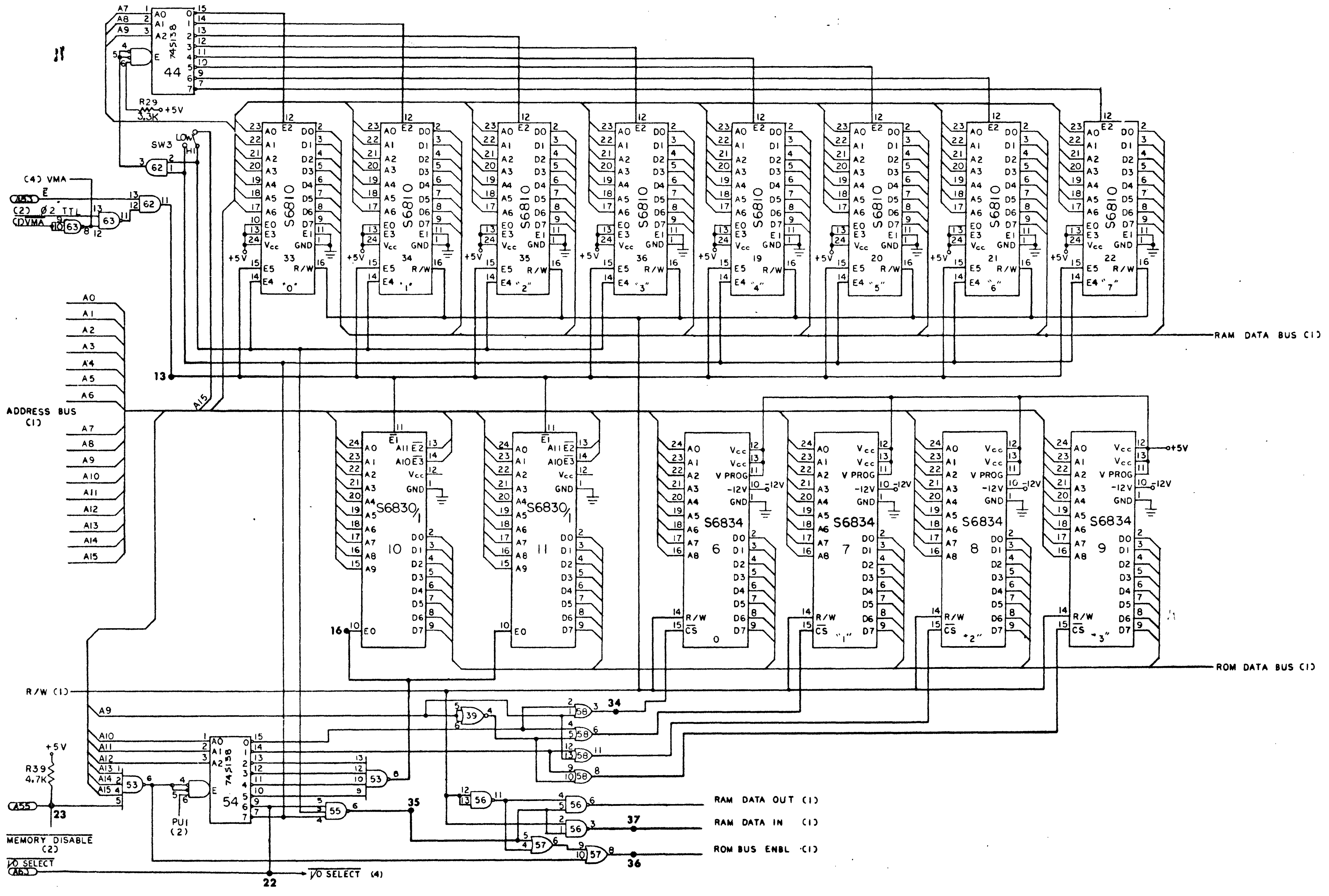


Figure 4. AMI S6800 Prototyping Board Schematic Diagram

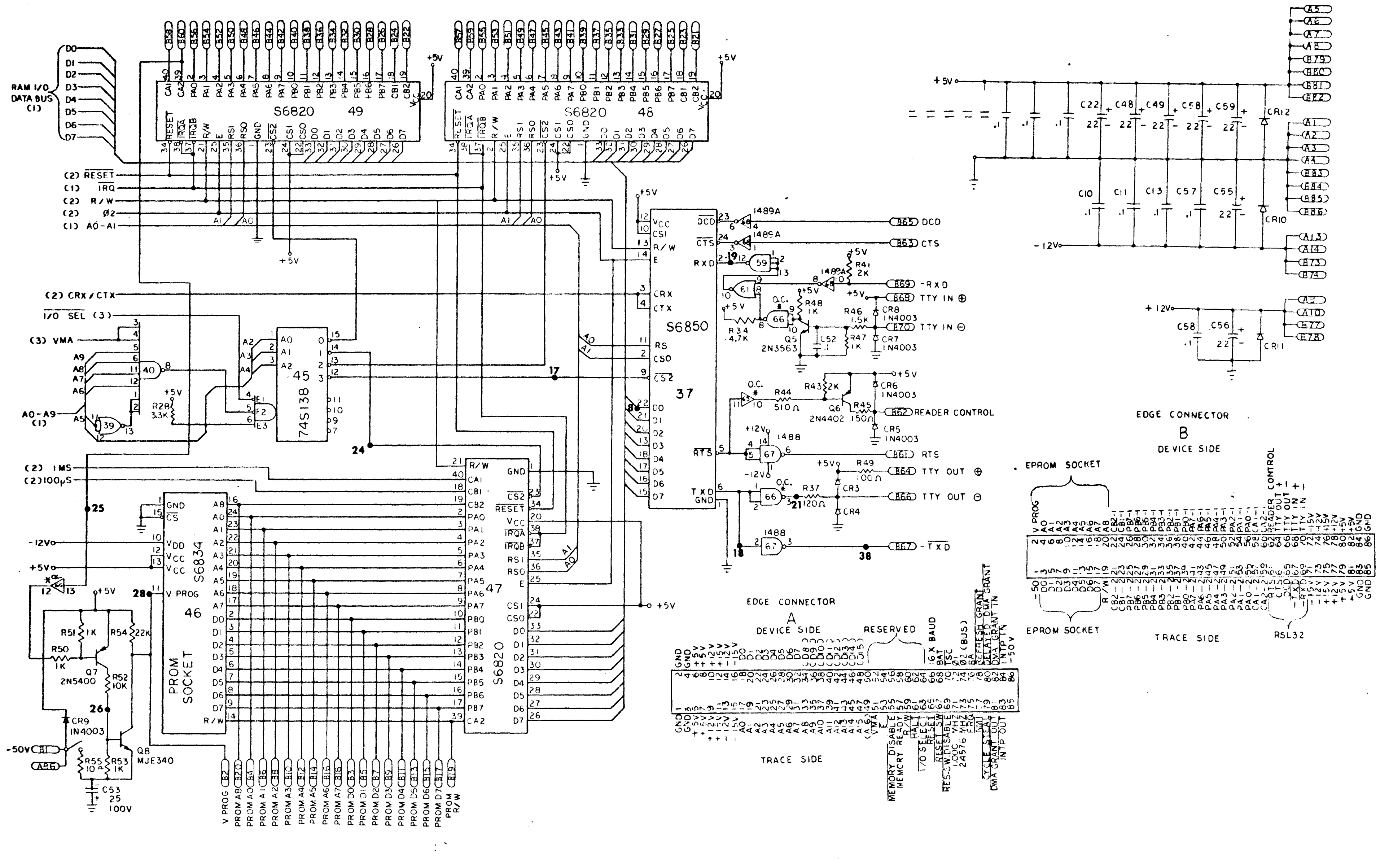


Figure 5. AMI Prototyping Board Schematic Diagram

EVK MICROCOMPUTER FUNCTIONAL DESCRIPTION

FUNCTIONAL CONFIGURATION

The following functional description is directed towards the fully populated EVK 300 Microcomputer board and is functionally applicable to the complete series of EVK boards limited only by the degree of board hardware-software population.

No attempt will be made to functionally describe the characteristics of AMI's Microcomputer IC chip set as this was undertaken in the first article entitled AMI 6800 Microcomputer Chip Set published last month in INTERFACE AGE. Instead, we will describe the general architecture of the EVK Microcomputer board and its general characteristics in order to provide an insight into EVK board utilization.

GENERAL ORGANIZATION

The EVK Microcomputer functional configuration is composed of the following major functional sections: MPU, clock, internal timer, memory, EPROM programmer, internal bus, expansion bus, I/O bus, I/O, and control logic sections. This functional interrelationship is shown in Fig. 1, EVK Microcomputer Functional Block Diagram while the detailed logic and circuit information is shown in Fig. 2, 3, 4, & 5, EVK Microcomputer Logic Diagrams.

MPU — The MPU is mechanized with AMI's S6800 Microprocessor chip. All MPU data address and control lines are buffered, and in addition are available at the board edge connector.

MPU TWO-PHASE CLOCK — The basic MPU two phase clock is derived from a 96S02 dual one-shot (IC12) connected either in a regenerative feedback loop or driven by a 1 MHz crystal controlled oscillator circuit (IC14). Switch #SW 2 is used to select either the one-shot regenerative feedback or the crystal oscillator mode of operation. Phase one and two timing is controlled by potentiometers connected to the one-shot RC timing networks and controls the phase pulse widths. These two phase additive pulse widths determine the MPU clock rate when the one-shot regenerative feedback configuration mode is connected. In this regenerative feedback mode, MPU clock frequency may be adjusted from 300 KHz up to 1 MHz. The phase timing outputs of the one-shots in both modes of operation drive 2N5771-2N5772 transistor amplifier circuits which in turn drive the two phase clocks of the MPU. In addition, both clock phases are buffered and available at the board edge connector. The fixed frequency 1 MHz crystal oscillator circuit output is also buffered and available at the board edge connector.

The two phase clock can be halted in either phase 1 or phase 2 for cycle-steal, DMA or slow memory applications. Phase 1 is halted (held HIGH) by driving the $\overline{\text{CYCLE STEAL}}$ control line LOW. Phase 2 is halted (held HIGH) by driving the $\overline{\text{MEMORY READY}}$ line low. Because the S6800 internal registers are dynamic and must be refreshed periodically $\overline{\text{CYCLE STEAL}}$ and $\overline{\text{MEMORY READY}}$ line outputs to the one-shots cannot be held LOW for more than 5 μs. This time limit protection, regardless of control input conditions, is provided by open collector 7407 (IC65) Hex non-inverting drivers, disconnect diodes and one-shot RC pull-up timing networks.

INTERNAL TIMER — A crystal controlled interval timer provides 100μs and 1 ms time periods for interrupting the MPU for real time clock applications. The 1 MHz crystal clock output drives a three decade divide-by-ten 74160 counters (IC50, 51, & 52) which in turn provide the 100μs and 1 ms time intervals. The 100 μs time interval pulse sets bit 7 of I/O address FBC7 via the S6820 PIA (IC47) while the 1 ms time interval pulse sets bit 7 of I/O address FBC5 via the S6820 PSA. These two time interval signals are used for timing EPROM programming.

MICROCOMPUTER BUS ARCHITECTURE — The EVK Microcomputer in essence has three sets of busses, namely the MPU bus, the Microcomputer bus and the on board memory-I/O bus. Each bus set consists of an 8-bit bidirectional data bus, a 16-bit unidirectional address bus and a control bus. The MPU bus is isolated from the Microcomputer bus in order to keep MPU signal loading to a minimum. The on board memory-I/O bus is isolated from the Microcomputer bus in order to assure that the on board memory and I/O devices do not load down the Microcomputer bus. As a result of this load isolation 40 ma drive current is available to drive external expansion hardware. The bus isolation buffers are non-inverting 3-state hex buffers (8T97). All of the controls to and from the S6800 are available at the board edge connector. This allows the user complete access and control of the MPU. Bus logic polarity is the same on all three busses (logic true = voltage high = "1"). The enable control signals to the MPU are always active. Control signals for the address bus are gated by the DMA GRANT line. The data bus is controlled by the DMA and R/W Lines.

MEMORY — The on board memory includes 1K bytes static RAM, 4K bytes ROM and 2K bytes EPROM.

MEMORY ADDRESS ASSIGNMENTS — Address assignments have been made such that all components on the card can run in the upper 8K bytes of memory. An address assignment map is shown in Figure 6.

Address decoding is made by use of three 74S138 one-of-eight decoders (IC 44, 45, 54). The first decoder (IC 54) selects one 1K-byte block of the upper eight 8K-bytes of memory. The output of this decoder is for RAM, I/O, ROM, or PROM enable lines. The second decoder (IC 44) selects one of eight RAM memory chips. The third (IC 45) selects I/O devices on the board.

A MEMORY DISABLE line is available at the Bus edge connector. This line, when LOW, deselects the first address decoder disabling all I/O and memory devices on the board. An I/O ENABLE line is derived from the first adress decoder and is available at the Bus edge connector. It must be noted that I/O ENABLE on the backplane is not valid when MEMORY DISABLE is LOW.

READ ONLY MEMORY — The Prototyping Board has assigned locations for two 1K byte S6830 ROMs and for four 512 x 8 S6834 EPROMs. The ROM circuits are designed such that the locations will also accept two 2K byte 16K ROMs (S6831). Thus, maximum memory allocation for ROM and EPROM is 6K bytes. The prototyping operating system program (PROTO) is assigned to the ROM with a starting address of F000.

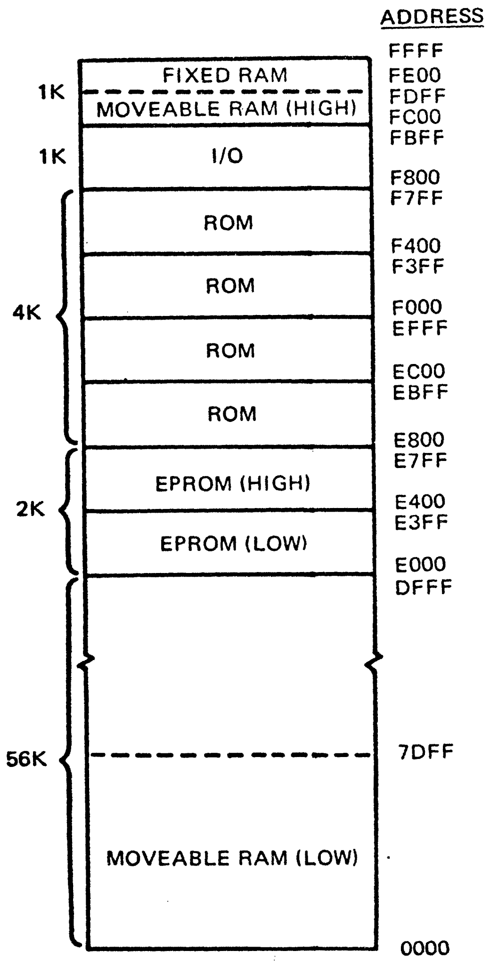


FIGURE 6. MEMORY ASSIGNMENT MAP FOR THE AMI PROTOTYPING BOARD

The four EPROM locations may contain any user program. Execution can start from beginning EPROM location either by selecting EPROM starting address of E000 in the restart switches or by branching to that address using the "G" command in the PROTO program.

RANDOM ACCESS MEMORY — The RAM is divided into two parts, 512 bytes fixed in the highest memory locations and 512 bytes of moveable memory.

Since the highest memory locations (FFFE, FFFF) are used for restart address, the address circuits disable the RAM using a memory disable line and force the 16 bit switch address on the data bus whenever a Reset occurs. This allows the user to vector to any address as his restart address.

The PROTO program assigns restart vectors for IRQ, NMI, and SWI whenever it is started (usually via Reset). It is therefore important to note that the user program must do the same thing if he does not use PROTO and restarts from a power down mode.

The stack pointer is assigned to address FF8F in PROTO. This allows the remaining RAM to be used as stack if so desired.

A switch option allows 512 bytes of RAM to be relocatable. When in the upper portion of memory, the RAM is assigned to addresses FC00 to FDFF making all 1K-bytes of RAM on the board contiguous (FC00 to FFFF). When in the lower portion of memory, the 512 bytes are addressed whenever A9 and A15 are not true (0000 — 01FF for example). It is thus recommended that RAM be assigned to the low address only if the user does not add other RAM to his development system.

I/O — On board I/O includes parallel PIA I/O ports and serial ACIA TTY and RS232C I/O ports.

PARALLEL I/O— Three S6820 PIA's give the user a wide range of I/O flexibility. The PIA's are assigned addresses as shown in Table 2. Interface pins of these devices are directly connected to the I/O edge connector. The CA2 pin for the PIA at addresses FBC4 is also connected to the VPROG input (pin 11) to the EPROM socket (IC 46) through a +5V to −50V driver. The user is cautioned to use this line such that it will not interfere with his I/O function if programming an EPROM. For example, if the CA2 line is connected to an external control function, this function may be erroneously activated while programming an EPROM.

TABLE 2. I/O ADDRESS ASSIGNMENT

I/O PORT	ADDRESS	ASSIGNMENT
S6850 ACIA		Serial I/O – TTY
	FBCE	Status/Read
	FBCF	Control/Write
S6820 PIA 1		Unassigned
	FBC8	Peripheral Register A
	FBC9	Control Register A
	FBCA	Peripheral Register B
	FBCB	Control Register B
S6820 PIA 2		Keyboard/Unassigned
	FBC0	Peripheral Register A
	FBC1	Control Register A
	FBC2	Peripheral Register B
	FBC3	Control Register B
S6830 PIA 3		PROM Burner
	FBC4	Peripheral Register A
	FBC5	Control Register A
	FBC6	Peripheral Register B
	FBC7	Control Register B

SERIAL I/O — One S6850 ACIA allows the system to communicate bi-directionally with serial data I/O peripherals such as a TTY. A baud rate generator generates all standard communication frequencies by switch selection. This frequency operates independently of the system clock so the MPU frequency can be changed without altering the I/O clock rate. See Table 3 for switch setting and associated frequencies. A 20 mA current loop interface and an RS-232 interface are both available at the I/O edge connector.

Address assignments for the ACIA are given in Table 3, "Bit Rate Generator Switch Settings."

EPROM PROGRAMMER — A unique feature of the Prototyping Board is its ability to program AMI S6834 EPROMs. A third PIA latches the address and data information for programming the EPROM. The EPROM socket programs only the S6834 EPROM, however, an adapter plug is available to also program the AMI S5204A EPROM. Except for the VPROG input, all address, chip select, R/W and data I/O pins on both EPROMs are completely TTL compatible and are driven directly from the PIA outputs. The outputs are also available on the I/O edge connector for convenience in using another EPROM programming socket.

TABLE 3. BIT RATE GENERATOR SWITCH SETTINGS,

0 = CLOSED, 1 = OPEN

SW POSITION				BIT RATE
4	3	2	1	
0	0	0	0	19,200 baud
0	0	0	1	0 baud
0	0	1	0	50 baud
0	0	1	1	75 baud
0	1	0	0	134.5 baud
0	1	0	1	200 baud
0	1	1	0	600 baud
0	1	1	1	2,400 baud
1	0	0	0	9,600 baud
1	0	0	1	4,800 baud
1	0	1	0	1,800 baud
1	0	1	1	1,200 baud
1	1	0	0	2,400 baud
1	1	0	1	300 baud
1	1	1	0	150 baud
1	1	1	1	110 baud

Programming is achieved by pulsing the VPROG pin with −50 volts through the CA2 line of the PIA at address FBC4. This line drives the transistor that gates the −50 volt source to the VPROG pin. The −50 volt source is switched ON or OFF via the VPROG switch.

CONTROL — The Microcomputer control section includes system reset logic, addressable reset logic and DMA control logic. In addition, an external logic circuit may be added to provide selection between RUN and single step modes.

RESET — The Reset circuit provides a timed reset for Power On Reset timing and for the Reset switch. The circuit is a timed oscillator which provides a 200 ms reset pulse.

RESTART — The starting address of an S6800 is FFFE/FFFF. The contents of these memory locations are put into the Program Counter register each time the MPU is reset. The Evaluation Board traps the FFE/FFFF addresses and puts the contents of the two 8-bit switch sets (IC 32, 43) on the data bus for each address and disabling memory, then gating the first set of switches to the Data Bus during FFFE time and the second set during FFFF time. The user is thus allowed to select any restart address by simply selecting a two byte address on the 16 bits of switch settings. The two DIP switches may be replaced with four hex thumbwheel switches mounted on a front panel and interconnected via a flat ribbon cable and DIP plug connectors providing front panel Hex restart control.

DMA — Three types of DAM implementation are possible on the Prototyping Board, a halt processor mode, a cycle steal mode and a multiplex mode. A switch selects these DMA modes. The switch must be in the DMA position for the multiplex DMA mode. A delayed clock gives the DMA GRANT line to the bus after the "Data Hold" time has passed for a multiplexed type of DMA operation. The control lines for the halt processor and cycle steal modes are available at the Bus edge connector.

RUN/HALT & SINGLE STEP EMBELLISHMENTS — A simple low cost three IC RUN/HALT-Single Step Instruction logic may be added external to the EVK board to provide these capabilities if required. Figure 7, RUN/HALT & Single Cycle Instruction Logic Diagram and Figure 8, Single Step Timing Diagram depicts this added logic mode.

POWER REQUIREMENTS

The EVK board is mechanized so that nominally only a +5 volt @ 3.5 amp power supply is required. A −12 volt supply is required when using S6834 EPROM ICs. In addition, a −50 supply is required when programming these EPROMs. The RS232C Interface requires both the +12V and −12V supplies for proper operation. The following is the total power and voltage level requirement for a complete operational EVK 300 Microcomputer board;

+5V @	4 Amps
−12V @	150 ma
+12V @	50 ma
−50V @	50 ma

SOFTWARE

The EVK 300 Prototyping Board Software is comprised of a TTY Operating Program (PROTO) and is supported by a ROM Subroutine Library (RS)[3].

PROTO— The EVK 300 is supplied with a prototyping operating system program (PROTO). The program resides in ROM with a starting address of FØØØ. The various routines within PROTO are called by entering via the TTY keyboard one of the commands. A command consists of one character command identifier followed by additional parameters, if needed, separated by blanks or commas. All commands end with a carriage return. Since no action is taken before the carriage return, an input line may be deleted by the use of the TTY ESCAPE key. The PROTO program operates on the following commands:

L	Load Memory from TTY paper tape (HEX Format)
P	Punch a Memory location to TTY paper tape (HEX Format)
S	Set (write) Memory to a given value
D	Display the contents of a memory location in HEX
G	Go to user program at specific address and begin program execution
R	Print contents of MPU C, B, A, X, P & S register on the TTY
B	Burn (program) an EPROM from Memory location indicated
V	Verify the contents of an EPROM with a specified memory location
I	Input (copies) contents of EPROM in the programming socket into memory.
M	Move a specific block of memory to a designated location
E	End of transmission (EOT) character terminates the record and punches EOT on paper tape.

The commands will operate on a single character op code plus address parameters from the TTY keyboard.

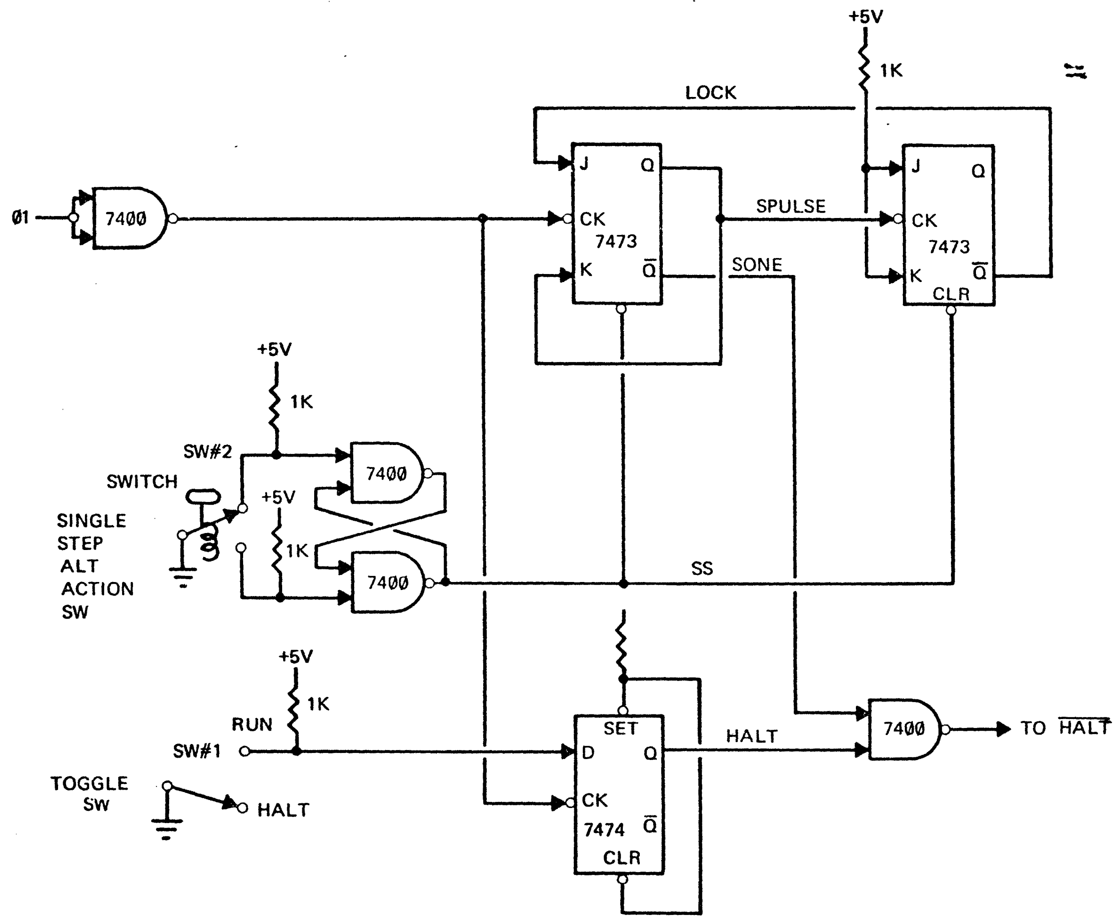


Figure 7. Run/Halt and Single Cycle Instruction Logic Diagram

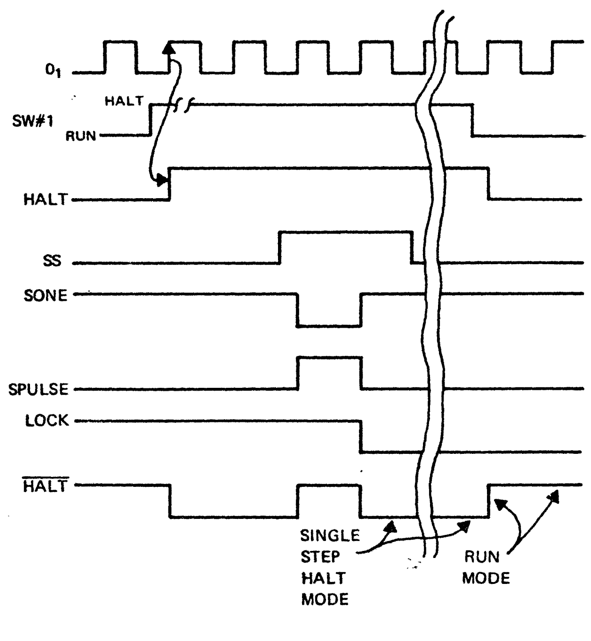


Figure 8. Single Step Timing Diagram

(RS)^3 SUBROUTINES — The 2K X 8 ROM provided with the PROTO prototyping system includes a set of (RS)^3 subroutines with a slightly different linkage from the standard (RS)^3 form, although the calling sequence is the same. In particular, the provision for additional subroutines in the of other (RS)^3 ROMs is limited to a total of 127 subroutines. The first additional (RS)^3 ROM address must be placed in RAM location FFF4 (which can be set via the Set Memory command or modified by an initialization code in a user program). Also, since it is incorporated into a larger program, the whole of which very nearly fills the 2K bytes of its ROM, the (RS)^3 part of the ROM does not start on an even page boundary, making it awkward for isolated use. However, the 24 subroutines included in this ROM are available to user program calls with the SWI calling sequence, as described.

The ROM Subroutine Library (RS)^3 operates on a single SWI (3F) command and a second byte of offset giving the S6800 an additional set of two-byte instructions. Specific subroutines (offsets) are as follows.

SUBROUTINE INDEX	MNEMONIC	FUNCTION
0	PUSHALL	All registers are pushed on to user stack.
1	POPALL	All registers on user stack are loaded into MPU.
2	TXAB	Contents of Index Register are transferred to A & B Accumulators.
3	TABX	Contents of A & B Accumulators are transferred to Index Register.
4	XABX	Contents of A & B Accumulators are exchanged with contents of Index Register.
5	PUSX	Contents of Index Register are pushed onto user stack.
6	PULX	Index Register is loaded with contents of user stack.
7	ADDXAB	Contents of Index Register are added to contents of A & B Accumulators. Sum is in A & B Accumulators.
8	ADDABX	Contents of A & B Accumulators are added to contents of Index Register. Sum is in Index Register.
9	ADDAX	Contents of Accumulator A are added to contents of Index Register. Sum is in Index Register.
10	ADDBX	Contents of Accumulator A are added to contents of Index Register. Sum is in Index Register.
11	SUBXAB	Contents of Index Register are subtracted from contents of A & B Accumulators. Remainder is in Accumulators A & B.
12	SUBABX	Contents of Accumulators are subtracted from contents of Index Register. Remainder is in Index Register.
13	SUBAX	Contents of Accumulator A are subtracted from contents of Index Register. Remainder is in Index Register.
14	SUBBX	Contents of Accumulator B are subtracted from contents of Index Register. Remainder is in Index REgister.
15	P2HEX	Two Hexidecimal Characters (one MPU byte) are printed on the TTY.
16	P4HEX	Four Hexidecimal Characters (two MPU bytes) are printed on the TTY.
17	PRINTA	ASCII Character designated is printed on TTY.
18	PMESS	Message designated is printed on TTY.
19	VALAN	Character (byte) is checked to see if it is a valid alpha/numeric character.
20	INPUTA	ASCII Character at TTY is input to MPU.
21	CONHB	ASCII Character string is scanned looking for a valid Hexidecimal number. Binary equivalent is returned in Accumulators A & B.
22	INDEX	Contents of Accumulator A are multiplied with the contents of Accumulator B and the product is added to the contents of the Index Register.
23	MUL8	Contents of Accumulator A are multiplied with the contents of Accumulator B. Product remains in both Accumulators.

S6800 MICRO ASSEMBLER/DISASSEMBLER (MA/D) — An optional ROM resident Micro Assembler/Disassembler is available for the EVK Microcomputer board at an additional cost of $30.00. Where this option is provided for those applications it may be desirable to debug programs using the mnemonic instruction codes instead of hexadecimal values. MA/D is designed to accomplish this by interfacing with a user via a keyboard and display (TTY or equivalent). The required 6800 environment must include:

Character in routine at location	0
Character out routine at location	3
No. nulls after carriage return at location	6
RAM at locations	7 - 78 $_{10}$

The I/O routines must transfer the characters in Register A and return with a RTS. It is expected that location 0 will just include a JMP to the actual character in routine, or, in the case of AMI's proto board:

```
00 SWI
01   FCB 20
02 RTS
```

The stack pointer must also be initialized before MA/D is entered. MA/D itself can execute from ROM, located anywhere in the system. MA/D may be started at its beginning address +2, in which case it will set up its environment for the AMI proto board.

After entering MA/D, the line length may be changed. The line length is in location 7 and is initially set to $(20)_{10}$=14 hex. The line buffer itself begins in location $(58)_{10}$=3A hex.

After displaying a header message MA/D prompts the user for a command by displaying MA/D's current location counter followed by a colon (:). The commands available to the user allow for disassembly of instructions in memory and assembly (mnemonic translation and operand insertion with relative offset computation) of instructions directly into memory.

MA/D is also very useful for writing short test programs. The instruction format for assembly is identical to the S6800 Assembler except:

1) operands must be in hexadecimal without the $, and no more than four digits long
2) no symbols can be defined or referenced
3) relative addresses are specified as absolute addresses, the offset is computed
4) in those instructions having **both** direct and extended addressing modes, extended addresses must have at least three digits. Thus,
```
   LDA A   10   assemblies as 96 10
   LDA A  010   assemblies as B6 00 10
```
5) in those instructions not having a direct addressing mode, the operand may be two or more digits. Thus,
```
   INC 10   assembles as 7C 00 10
```
6) an operand may be a single hex digit only if the op code indicates an A or B register, or immediate mode addressing. Thus,
```
   INC  01           INC  1
   LDA  A  1         LDX  1
   LDX  #1
```
   —are legal—   —are not—

   (This makes it easier to distinguish between, for instance, INC A and INC 0A.)
7) Anywhere a number is used, the construction 'character may be used instead, and is equivalent to the ASCII code for the character.

MA/D Command Summary

@newloc @ newloc	The @ sign followed (immediately or with blank separator) by a hexadecimal address initializes the current location counter to the new address. MA/D automatically updates the location counter as instructions are assembled or disassembled.
$count $ count	The $ sign followed by a one or two digit (hexadecimal) count results in the disassembly of "count" instructions. Zero = infinity.
!address ! address	The exclamation mark followed by an address causes MA/D to call a subroutine at the given address. If the subroutine returns with the carry flag set, MA/D will print "????".
!	Exclamation mark with no address given causes MA/D to call the subroutine starting at the current location.
"string	Assembles the ASCII characters following the double quote mark into successive bytes of memory starting at the current location. The current location is updated.
xx x 'character	A one or two digit hexadecimal number is placed into the current location, and the current location is incremented by one. A single quote mark followed by a single character causes the ASCII code for that character to be placed in the current location. This command may appear several times on the same line, the numbers or quoted characters separated by spaces or commas.
&address,count & address,count &address count & address count	Ampersand followed by a hexadecimal address and count (from 1 to 4 digits each) causes "count" bytes to be moved from "address" to the current location. On completion, the current location is incremented by "count".
<RETURN>	Carriage return is equivalent to $01, disassemble a single instruction.

The commands to MA/D are buffered and not processed until the ‹RETURN› key is depressed. The ‹BACKSPACE› key can be used to delete the last character input. If errors are detected on user input the line is ignored, ???? is displayed, and another prompt is issued.

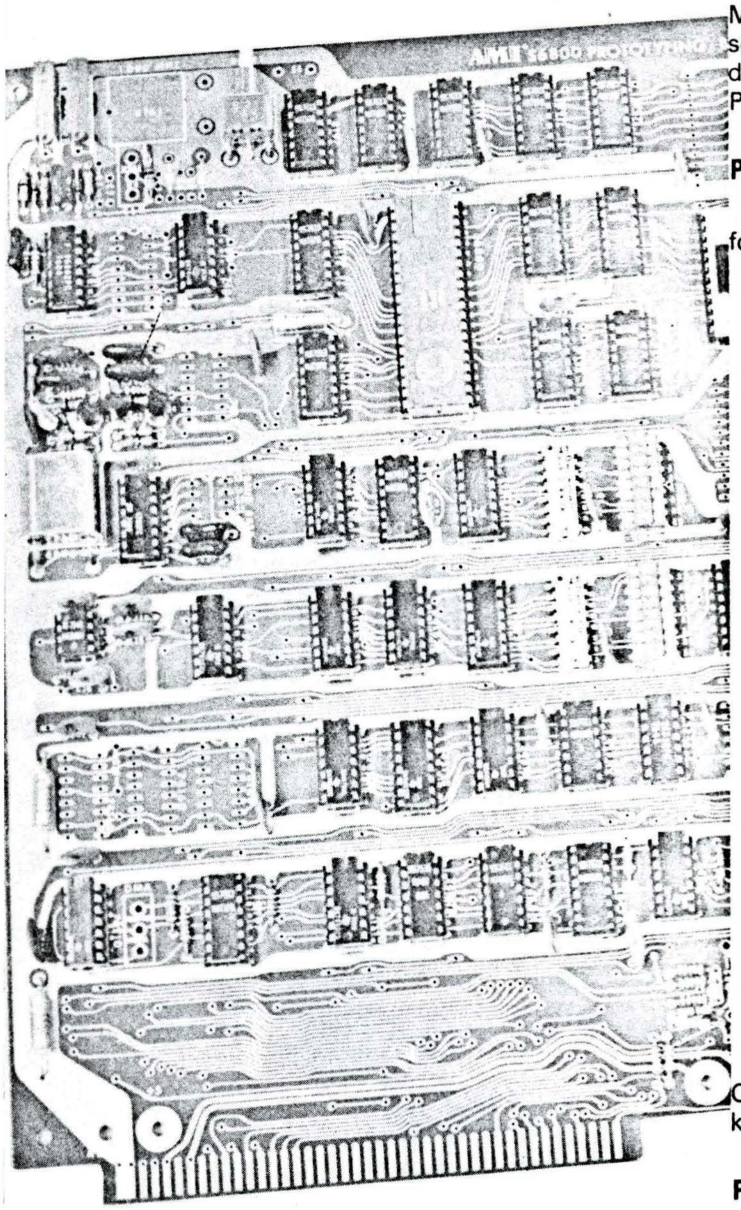
The default command is "assemble" and MA/D, if not recognizing the input as one of the following commands, generates the machine code for the instruction mnemonic.

Next month we will publish the complete PROTO Assembly Listing for the EVK Microcomputer board.

```
>
>G E002
A.M.I. 6800 MICRO ASSEMBLER/DISASSEMBLER — 1.0
(C) 1976, A.M.I.
002A:@80
0080:"THIS IS LOOP NO.
0090:"0000
0094:04
0095:LDA A 93
0097:INC A
0098:STA A 93
009A:CMP A #3A
009C:BNE 110
009E:LDA A '0
00A0:@9E
009E:
009E-> 96 LDA A 30
00A0:@9E
009E:LDA A #'0
00A0:STA A 93
00A2:LDA A 92
00A4:INC A
00A5:STA A 92
00A7:BRA 110
00A9:@110
0110:LDX #0080
0113:LDA A 00,X
0115:CMP A #04
0117:BEQ 120
0119:JSR E003
011C:@119
0119:JSR 0003
011C:INX
011D:BRA 113
011F:NOP
0120:LDA A #0D
0122:JSR 0003
0125:LDA A $0A????
0125:LDA A #0A
0127:JSR 0003
012A:JMP 095
012D:@95
0095:$3
0095-> 96 LDA A 93
0097-> 4C INC A
0098-> 97 STA A 93
009A:!95THIS IS LOOP NO.0001
THIS IS LOOP NO.0002
THIS IS LOOP NO.0003
THIS IS LOOP NO.0004
THIS IS LOOP NO.0005
THIS IS LOOP NO.0006
THIS IS LOOP NO.000
>
>
```

AMI's EVK SERIES MICROCOMPUTER PROTOTYPING BOARDS

By Robert A. Stevens



INTRODUCTION

This article is Part Three of a series on the EVK Microcomputer hardware, firmware and supporting software. This month's subject covers the ROM resident Prototyping TTY MONITOR Operating System, PROTO.

PROTO SOFTWARE

The resident PROTO software program includes the following commands:

- **L** **LOAD HEX** paper tape program into RAM memory
- **P** **PUNCH HEX** paper tape from memory
- **S** **SET** (write) specified data string characters into consecutive memory locations
- **D** **DISPLAY** (prints) in HEX to TTY contents of specified memory locations
- **G** **GO TO** user program at specified address and execute
- **R** **PRINTS** contents of MPU register (C, B, A, X, P & S) on TTY at time the user's program was last interrupted
- **B** **BURN** (copies) the contents of specified memory into the EPROM in the programming socket
- **V** **VERIFY** (compares) contents of specified memory with EPROM or ROM in the programming socket
- **I** **INPUT** (copies) contents of the EPROM or ROM in the programming socket into specified RAM memory locations
- **M** **MOVE** (copies) contents of memory block from specified location to designated RAM memory location
- **E** **END** of transmission (EOT) character terminates the end of punch paper tape record and punches EOT on paper tape.

The commands will operate on a single character OP CODE plus address parameters from the TTY keyboard.

PROTO COMMAND DESCRIPTIONS

The EVK 300 board will be supplied with a prototyping operating system program (PROTO). The program resides in ROM with a starting address of F000. The various routines within PROTO are called by entering via the TTY keyboard one of the commands described in the following paragraphs. A command consists of one character command identifier followed by additional parameters, if needed, separated by blanks or commas. All commands end with a carriage

Good Software and Support are to a computer as the driver is to his car. One without the other and you have a magnificent paperweight.

return. Since no action is taken before the carriage return, an input line may be deleted by the use of the TTY ESCAPE key.

L, ADDL, ADDH, OFFSET

The Load tape command loads data from a hex formatted tape (see paragraph on 6800 HEX tape format at end of article) into the user's memory between ADDL and ADDH, inclusive. The OFFSET is added to the memory address specified on the tape to form the actual memory starting address for the data stored. If a byte to be stored into memory has an address outside of the range ADDL, ADDH, it is not entered into memory, but a Delete character (H'FF) is transmitted to the terminal.

Example: L 0100 02FF FFFA

The address range in the L command is optional, and if omitted is assumed to be the full range of memory (0000-FFFF). The offset parameter is also optional, and if omitted is assumed to be zero (0000). Thus the L command with no parameters loads the tape into the memory locations specified on the tape with no offset. The offset value in the L command is a two's complement signed number, entered in unsigned hexadecimal. For example, an offset of −6 is entered as FFFA.

If an attempt is made to load non-existent memory, or ROM, the loading operation will terminate, typing out the address and the message "BAD ADR."

In operating the Load command, PROTO turns on the tape reader and scans the tape for the first ASCII "S," which indicates start of record. It is not necessary to position the tape at the first record of a tape file since each record contains its own starting address.

PROTO will load data records until it encounters an end of file (EOF) record or a tape error (Check Sum or illegal character). When PROTO reads a header record (start of record and address), it translates the header into ASCII characters and prints the result. The Check Sum is the binary sum of all characters in the block.

PROTO does not list the tape contents as the tape is being read.

When PROTO encounters an end of file record or a tape error, it turns off the reader and prints "EOF" or "CKSM ERR" respectively.

P, ADDL, ADDH, OFFSET

The Punch hex format command causes PROTO to punch on the TTY paper tape the contents of memory between ADDL and ADDH, inclusive. Each record is punched with a four-digit hex address of the starting byte of the record. This address is derived from the memory address of the byte being punched, plus the offset value, OFFSET. The offset is optional, and if omitted is assumed to be zero.

All data records are punched in hex format. Records using this command (except the last record) contain 16 bytes of data plus the start code, byte count, address, and the checksum.

The P command does not cause an EOF record to be punched so that several disjoint blocks of memory can be combined on one tape file.

Example: P F000 F07F 0F00

S, ADDR, BYTE1, BYTE2, ———, BYTEN

The Set memory command writes the 8-bit data words specified by BYTE1 to BYTEN into consecutive memory locations starting at ADD.

If ADD has more than 4 (hexadecimal) characters or if any of the data bytes have more than 2 characters each, only the last 4 or 2 characters are used respectively.

Example:S 0000 86 05 97 28

Memory locations at 0000 through 0003 are loaded as shown.

D, ADDL, ADDH

The Display memory command prints the contents of memory between ADDL and ADDH, inclusive, in hex format. Up to sixteen bytes per line are printed, preceded by the hexadecimal address of the first byte of the line. A carriage return is forced after a byte having a low order digit of F in its memory address is printed.

Example: D FC00 FCIF

Two lines of memory contents are printed as follows:

```
FC00 00 01 02 03 04 . . . 0E0F
FC10 10 11 12 13 14 . . . 1E1F
```

G, ADDR

The Go command starts execution of the user program at the address specified by the input parameter. To insure that all registers contain the same information they held before the user program was interrupted, PROTO pushes into the stack the copy of the user registers that it keeps at locations FFEB—FFF3 (CC, B, A, X, P, S) then executes an RTI instruction. The user can change the initial values of the registers by changing the contents of these locations.

Example: G 300

Program will branch to address 0300 and start execution from that point.

R

The Registers command prints the contents of memory locations FFEF—FFF3 which contain the values that were in the user's C, B, A, X, P, and S registers (in that order) when the user's program was last interrupted.

B, ADDL, ADDH, ROMAD

The Burn command copies the contents of user memory into the EPROM in the programming socket, beginning with memory location ADDL through ADDH, inclusive, to EPROM locations beginning with address ROMAD. Each byte is burned in with 20 3-ms pulses of −50V on the V_{PROG} pin (pin 11) of the EPROM. Before attempting to write into the EPROM, the contents of the EPROM are compared with the user memory data byte to verify that the EPROM will take the byte (PROTO will not attempt to program a EPROM location to logic LOW which already contains logic HIGH). After the 20 pulses, the new contents of the EPROM are verified against the memory byte to be sure the data was indeed written. If the byte did not program, a NAK code is typed out on the terminal, and another try is made, up to a maximum of three tries.

If the preverify encounters a EPROM location containing HIGHs where the memory byte has zeros, PROTO will type out the memory address, the memory byte in binary, the EPROM byte in binary, and the EPROM address (if different from the memory address), then stop. If after attempting to write data into the EPROM, the data does not program, or erroneous bits show up, a similar display occurs for the failing location, with the additional message "BAD ADR" typed on the same line.

The EPROM address ROMAD is optional, and if omitted, ADDL is used, with only the least significant nine bits of the address being used. If the address range ADDL, ADDH is omitted, the 512 bytes beginning at FC00 are used, and the EPROM is checked to insure it contains all LOWs before any locations are written. If not, four question marks are typed and the B command is aborted.

V, ADDL, ADDH, ROMAD

The Verify command compares user memory between ADDL and ADDH, inclusive, with the corresponding locations in the EPROM in the prgramming socket, beginning with EPROM address ROMAD. Each location that does not match is typed out in the following format:

```
aaaa mmmmmmmm pppppppp rrrr
```

where "aaaa" represents the user memory address, "mmmmmmmm" represents the memory byte, in binary, and "rrrr" represents the EPROM address, if different from the memory address (in the low nine bits). Nothing is typed for matching locations. The typeout may be aborted by typing an ESC key during the typeout.

If the ROMAD parameter is omitted, ADDL is assumed. If no parameters are supplied in the command, the whole EPROM is compared to the contents of FC00 — FDFF.

I, ADDL, ADDH, ROMAD

The Input command copies the contents of an EPROM in the programming socket into memory beginning at the address ADDL through ADDH, inclusive, from the EPROM address ROMAD. If ROMAD is omitted, ADDL is assumed. If no parameters are supplied, the entire EPROM is copied into the RAM area, FC00 — FDFF. An attempt to copy an EPROM into non-existent memory will abort the command with the message "BAD ADR."

M, ADDL, ADDH, DEST

The Move command copies memory from the range ADDL — ADDH, inclusive, to the RAM locations starting at DEST. This copy begins at the lower address, so if DEST lies within the range ADDL — ADDH, some of the original data will be lost, and other parts will be duplicated.

E

The End of Transmission command is used to cause an EOT character to be punched on the paper tape. After a field has been punched, an EOT will terminate the record and punch a trailer tape. When reading a record, the reader will stop at the EOT character. If no EOT character is present, the reader must be manually turned off and the Reset switch must be pressed to enter the operating system program.

THE SUBROUTINE ROM

Many of the monitor's functions are accomplished with the help of the Re-Entrant Self-Relative Subroutine ROMs $(RS)^3$. This standard ROM, which can be considered a software extension to the 6800 instruction set, is also available to be used by the user both on the prototype board and in his final production system. The user can call one of the 25 $(RS)^3$ subroutines with an SWI instruction followed by the number of the desired subroutine.

The user should be aware of the fact that the $(RS)^3$ pushes from 7 to 10 bytes of data onto the stack, depending upon which subroutines are called. This means that if the user calls $(RS)^3$ routines, he must make sure that the necessary memory space is available for stack expansion.

Since PROTO assigns its own stack area, the user need not be concerned about how $(RS)^3$ is used.

INTERRUPTS

Of the four available interrupt vectors, IRQ, RESET and SWI are used by PROTO while NMI is left for the user. The vectors are in RAM (except for RESET which is switch controlled) so the user writing his own program can completely control the system.

The upper memory locations are RAM. If the user expects either NMI or IRQ interrupts to occur, he must initialize the vector addresses to the starting address of the IRQ and NMI handler routines.

PROTO must have control of the RESET vector so that the RESET switch on the Prototyping Board can return program control to PROTO at any time.

The reset routine copies the contents of the B, A, X, CC, and S registers into a fixed area of memory. This means that the program can be aborted at any time by using the reset switch while still saving all the registers except the program counter. Unfortunately, the contents of the program counter are lost.

It is possible for the user to use the NMI interrupt to abort a program execution without losing the contents of the P and C registers. This condition is automatically set in the NMI handling routine when PROTO is called. This interrupt vector will cause the contents of the user's registers to be printed when the $\overline{\text{NMI}}$ line goes low.

Since the SWI instruction is used to call subroutines between 00 and H'18 from (RS)^3 the user is somewhat limited in the ways he can use SWI instructions. However, he can access an SWI handler routine in his own program by an SWI instruction followed by a byte containing the decimal number less than H'80 but greater than H'19 < n < H'80 sequence, PROTO passes control at address FFF4. If the user expects to access his own SWI routine and use PROTO, he must use the Set Memory command to store the address of this routine at locations FFF4 and FFF5.

PROTO makes sure that the user's SWI routine is entered from the stack with all registers containing the same information that they would hold if the routine were entered directly through the SWI vector.

BREAKPOINTS

Breakpoints allow the user to halt his program and examine the contents of the internal registers. PROTO provides two types of breakpoints. In this system, breakpoints are actually debugging routines that can be called from the user's program just like (RS)^3 routines.

Each breakpoint requires a two byte calling sequence: and SWI instruction followed by a number.

Breakpoints may be inserted either by reassembling the program with the extra SWI instructions added or the Set Memory command may be used to replace parts of the code with SWI instructions. Note that the second method is not satisfactory for the snapshot option (described below) since the replaced code must be restored before execution can be continued. When using the second method, the user must make sure that he replaces the first two bytes of an instruction. If the SWI replaces the second or third byte of an instruction, it may be interpreted as an address rather than an opcode.

The different types of breakpoints are:

1. Print registers (SWI, H'80)
2. Snapshot (SWI, H'81)

The sequence SWI, H'80 saves the user's registers at the vector stored in FFF4 — FFF5, prints their contents (in the order CC BB AA XXXX PPPP SSSS), then returns control to PROTO.

The sequence SWI, H'81 prints out the contents of the user's registers then continues executing the user's program starting at the address following the byte containing the number H'81. Note that if this address does not contain a valid opcode, unpredictable results will occur.

6800 PAPER TAPE HEX FORMAT

The AMI 6800 Hex Tape format provides a compact representation of binary data patterns for transmission using ASCII communication terminals.

The Hex tape is organized into data records with each record containing information in the same format. The record information consists of type, length, address, data and checksum. All records begin with an 'S' character for start of record identification. All information on the tape which is not between a start of record and the checksum is ignored.

TAPE FORMAT

ASCII Character	Description
1	Start of record (S)
2	Type of record 0 — Header record 1 — Data record 9 — End of file record
3—4	Byte Count Since each data byte is represented as two hex characters, the byte count must be multiplied by two to get the number

	of characters to the end of the record. (This includes checksum and address data.)
5, 6, 7, 8	Address Value The memory location where this record is to be stored.
9,....,N	Data Each data byte is represented by two hex characters.
N+1, N+2	Checksum The one's complement of the additive summation (without carry) of the data bytes, the address, and the byte count.

Example Data Record

Memory Contents

Address	**Data**
A000	10
A001	1A
A002	20
A003	2A

Data Record Contents

Character		**Tape**			
1	Start of record	53	S		
2	Type of record	31	1		
3	Byte count	30	0		
4		37	7		
5		41	A		
6		30	0		
7	Address	30	0		
8		30	0		
9	Data byte 1	31	1	Checksum	
10		30	0		
11	Data byte 2	31	1		Byte Count *2
12		41	A		
13	Data byte 3	32	2		
14		30	0		
15	Data byte 4	32	2		
16		41	A		
17	Checksum	38	8		
18		34	4		

The format for all hex tape records is diagrammed below.

Character		**Header Record**		**Data Record**		**End-of-File Record**	
1	Start of Record	53	S	53	S	53	S
2	Type of Record	30	0	31	1	39	9
3	Byte Count	31	12	31	16	30	03
4		32		36		33	
5		30		31		30	
6	Address	30	0000	31	1100	30	0000
7	(if any)	30		30		30	
8		30		30		30	
9	Data	34		39	98	46	FC
10		38		38		43	(Checksum)
•		34		30	02		
•		34		32			
•		35					
•		32					
•				41	A8 (Checksum)		
•				48			
N	Checksum	39	9E				
[illegible]		45					

SEE MICROCOMPUTER SOFTWARE DEPOSITORY PROGRAM INDEX FOR COPIES OF THIS PROGRAM.

PROTO PROGRAM

```
PAGE  1  PROTO   01/09/76  9:22  PROTO

STMT  LOC   OBJECT  M  SOURCE STATEMENT

                          TITLE  PROTO
                          OPT    LSKP
                 ************************************************
                 *
                 * PROTOTYPE BOARD MONITOR PROGRAM
                 *
                 * VERSION 2.0  01/08/76
                 *
                 * COPYRIGHT 1976 BY AMERICAN MICROSYSTEMS INC.
                 *
                 *
                 ************************************************
                 *
                 * DEFINITIONS
                 *
            FBCE A ACIAC    EQU    $FBCE          ACIA CONTROL REG
            FBCF A ACIAD    EQU    $FBCF          ACIA DATA REG
            FBCE A ACIAS    EQU    $FBCE          ACIA STATUS REG
            0020 A BLANK    EQU    $20            BLANK CHAR
            000D A CR       EQU    $0D            CARRIAGE RET CHAR
            0018 A ESC      EQU    $18            ABORT CHAR
            0004 A EOT      EQU    $04            END OF MSG TO BE PRINTED
            FFFF A LAST     EQU    $FFFF          HIGHEST ROM ADDRESS
            000A A LF       EQU    $0A            LINE FEED
            007F A RUBOUT   EQU    $7F
                 *
                 *   EXTERNALS FOR RSRSH AND PROM BURNER
                 *
                          DEF    MONENT,GETRNG,NXTADR,PXISTS,RNGERR,PBADR
                          DEF    PCRLF,OUTCH,PSPACE,SETMEM,ABORT
                          DEF    PROMAD,ADR,ADDL,ADDH,COUNT,MONITR
                          REF    RSRSH,BURN,MOVE,READ,VFY,PINIT
                 *
                 *RSRSH ROUTINE DEFINITIONS:
                 *
            000B A SUBXAB   EQU    11             SUBTRACT X FROM A,B
            0008 A ADDABX   EQU    8              ADD A,B TO X
            0012 A PMSG     EQU    18             PRINT MSG
            000F A P2HEX    EQU    15             PRINT BYTE AS 2 HEX CHARS
            0010 A P4HEX    EQU    16             PRINT WORD AS 4 HEX CHARS
            0015 A CONHB    EQU    21             CONVERT HEX TO BINARY
            0011 A PUTA     EQU    17             OUTPUT TO ACIA
            0014 A GETA     EQU    20             INPUT FROM TTY
            0013 A ALPNUM   EQU    19             TEST FOR ALPHANUMERIC
            0009 A PRIXD    EQU    9              CONV. X TO DEC. & PRINT
                 *
                 *'SUBR IS A MACRO TO CALL RSRSH ROUTINES
                 *
                 SUBR     MACRO  PARAM
                        SWI
                        BYTE  PARAM
                          MEND
                 *
                 ************************************************
                 *
                 * MONITOR RAM
                 *
FF90                      ORG    $FFFE-110      ***CHANGE IF RAM USAGE CHANGES
            FF90 A BASE     EQU    *              BASE ADR USED WITH INDEX OPS
            FF8F A BOS      EQU    *-1            BOTTOM OF MONITOR STACK
                 *
FF90        0048   BUF      RMB    72             LINE OF TTY INPUT
                 *
            FFD8 A PROMAD   EQU    *              ADDRESS IN PROM
FFD8        0002   OFFSET   RMB    2              OFFSET FOR LOADER/PUNCH
FFDA        0002   ADR      RMB    2              PARAM. ENTERED BY USER
FFDC        0002   ADDL     RMB    2
FFDE        0002   ADDH     RMB    2
FFE0        0002   BUFPTR   RMB    2              POINTER TO LAST CHAR SCANNED
FFE2        0001   RECTYP   RMB    1              TAPE RECORD TYPE
FFE3        0001   COUNT    RMB    1              COUNT FIELD FROM TAPE
FFE4        0001   CKSM     RMB    1              CALCULATED CKSM
FFE5        0002   SAVESP   RMB    2              TEMP STORAGE FOR S REG
FFE7        0002   SAVEX    RMB    2              TEMP STORAGE FOR X REG
FFE9        0001   ECHO     RMB    1              1=ECHO TTY, 0=NO ECHO
FFEA        0001   TCOUNT   RMB    1              TEMP LOC FOR COUNT
                 * USER REGISTERS
FFEB        0001   CREG     RMB    1
FFEC        0001   BREG     RMB    1
FFED        0001   AREG     RMB    1
FFEE        0002   XREG     RMB    2
FFF0        0002   PREG     RMB    2
FFF2        0002   SREG     RMB    2
                 *
FFF4        0002   USWI     RMB    2              USER SWI VECTOR (MAY NOT BE IMPLEMENTED)
FFF6        0002   ACIAI    RMB    2              INDIRECT POINTER TO ACIA FOR RSRSR
FFF8        0002   IRQVEC   RMB    2              INTERRUPT REQUEST VECTOR
FFFA        0002   SWIVEC   RMB    2              SOFTWARE INTERRUPT VECTOR
FFFC        0002   NMIVEC   RMB    2              NON-MASKABLE INTERRUPT VECTOR
                 ************************************************
                 *
                 *  *** MONITOR ENTRY VECTOR ***
                 *
                 * RESTART INTERRUPT HANDLER
                 * INTERRUPT BREAK HANDLER
                 *
                 ************************************************
0000                      ISEC
            0000 I START    EQU    *              RESET INTERRUPT HANDLER
0000  20 05                 BRA    START1
            0002 I BREAK    EQU    *              BREAK ON INTERRUPT ROUTINE
0002  7E 0087 I             JMP    BREAK1
0005        FBCE A ACIAA    WORD   ACIAC          POINTER TO ACIA
                 *
            0007 I START1   EQU    *
0007  36                    PSH A                 SAVE A REG IF STACK EXISTS
0008  07                    TPA                   SAVE CONDITION CODES
0009  B7 FFEB A             STA A  CREG
000C  32                    PUL A
000D  B7 FFED A             STA A  AREG           SAVE CURRENT VALUE OF REGS
0010  F7 FFEC A             STA B  BREG
0013  FF FFEE A             STX    XREG           SAVE X
0016  BF FFF2 A             STS    SREG           SAVE SP
0019  8E FF8F A             LDS    #BOS           INIT. SREG TO MON. STPCK
001C  CE 0002 I             LDX    #BREAK         BREAKPOINT ROUTINE
001F  FF FFFC A             STX    NMIVEC         STORE IN INTERRUPT VECTORS
0022  FF FFF8 A             STX    IRQVEC
0025  CE 00D1 I             LDX    #SWI30
0028  FF FFF4 A             STX    USWI
002B  CE 00BE I             LDX    #SWIHAN        SOFTWARE INTERRUPT HANDLER
002E  FF FFFA A             STX    SWIVEC
0031  CE 0005 I             LDX    #ACIAA         SET UP ACIA PTR
0034  FF FFF6 A             STX    ACIAI
0037  86 03                 LDA A  #3             RESET ACIA
0039  B7 FBCE A             STA A  ACIAS
                 *
003C  86 01                 LDA A  #1             SET ACIA CR
003E  B7 FBCE A             STA A  ACIAC
                 * PRINT CR,LF, & RETURN TO MONITOR
                 *
            0041 I MONENT   EQU    *
            0041 I MONEN1   EQU    *
0041  BD 0005 R             JSR    PINIT
0044  BD 0304 I             JSR    PCRLF
                 *
                 ************************************************
                 *
                 *
                 * MONITOR ENTRY POINT
                 *
                 ************************************************
            0047 I MONITR   EQU    *
0047  8E FF8F A             LDS    #BOS           INIT MON. STACK
004A  BD 0396 I             JSR    RDROFF         TURN OFF READER
```

```
STMT  LOC   OBJECT  M  SOURCE STATEMENT

004D  B6 FBCF A             LDA A  ACIAD          DUMP TTY INPUT DATA
0050  86 3E                 LDA A  #'>            PROMPT USER
0052  BD 0200 I             JSR    OUTCH
                 *
                 * READ TTY LINE (BUFPTR)
                 *  STORE TTY INPUT IN BUF UNTIL CR IS HIT
                 *
0055  CE FF90 A             LDX    #BUF           INITIALIZE BUFPTR
0058  FF FFE0 A             STX    BUFPTR
005B  0D                    SEC                   SET ECHO FLAG
005C  79 FFE9 A             ROL    ECHO
                 *BEGIN UNTIL LOOP
005F  8C FFD7 A  RT10       CPX    #BUF+71        TEST FOR BUF OVERFLOW
0062  26 02                 BNE    RT20           NO OVERFLOW
0064  20 47                 BRA    ABORT
0066  BD 0400 I  RT20       JSR    WAITTY         READ NEXT CHAR
0069  A7 00      RT30       STA A  0,X            INSERT CHAR INTO BUF
006B  08                    INX                   INC BUFPTR
                 * WHILE CONDITION :
006C  81 0D      RT90       CMP A  #CR            CARRIAGE RETURN ?
006E  26 EF                 BNE    RT10           NO, CONTINUE LOOP
                 *END OF LOOP
                 *
                 * DECODE 1 CHAR COMMAND
                 *   COMPARE CHAR WITH TABLE OF VALID CHARS FOLLOWED BY
                 *   ADDRESSES OF APPROPRIATE ROUTINES.
                 *
0070  BD 0365 I             JSR    PXISTS         GET 1ST CHAR
0073  08                    INX                   INC BUFPTR
0074  FF FFE0 A             STX    BUFPTR
0077  CE 008C I             LDX    #CTABLE        START OF TABLE
                 * BEGIN LOOP
007A  A1 00      DLOOP      CMP A  0,X            COMPARE
007C  26 04                 BNE    DL10
                 * FOUND CHAR.  GET ADDRESS IMMEDIATELY FOLLOWING CHAR.
007E  EE 01                 LDX    1,X
0080  6E 00                 JMP    0,X            GO TO PROPER ROUTINE
                 * NO COMPARE.  MOVE TO NEXT CHAR.
0082  08         DL10       INX
0083  08                    INX
0084  08                    INX
0085  8C 00AD I             CPX    #CTEND         END OF TABLE?
0088  26 F0                 BNE    DLOOP          NO. REPEAT
                 * END LOOP.
008A  20 21                 BRA    ABORT          NOT IN TABLE.
                 *
            0047 I MONEND   EQU    MONITR
                 *
                 *
                 * CTABLE:  TABLE OF VALID 1 CHARACTER COMMANDS.
                 *       EACH ENTRY CONSISTS OF 3 BYTES.  BYTE 1
                 *    CONTAINS THE ASCII CHAR.  BYTES 2,3 CONTAIN THE
                 *    ADDRESS OF THE APPROPRIATE ROUTINE.
                 *
            008C I CTABLE   EQU    *
008C        4C                BYTE   'L
008D        01AD I            WORD   LOAD
008F        47                BYTE   'G
0090        019F I            WORD   GO
0092        50                BYTE   'P
0093        0308 I            WORD   PUNCH
0095        42                BYTE   'B
0096        0001 R            WORD   BURN
0098        4D                BYTE   'M
0099        0002 R            WORD   MOVE
009B        56                BYTE   'V
009C        0004 R            WORD   VFY
009E        49                BYTE   'I
009F        0003 R            WORD   READ
00A1        53                BYTE   'S
00A2        03E3 I            WORD   SM
00A4        44                BYTE   'D
00A5        0136 I            WORD   DM
00A7        52                BYTE   'R
00A8        00EE I            WORD   PREGS
00AA        45                BYTE   'E
00AB        015D I            WORD   EOF
            00AD I CTEND    EQU    *
                 *
                 ************************************************
                 *
                 * ABORT
                 *
                 ************************************************
                 *
            00AD I ABORT    EQU    *
            00AD I BADINP   EQU    *
00AD  CE 0273 I             LDX    #MQUES         PRINT ????
                 *
                 * PRINT MSG AND RETURN TO MONITOR
                 *
            00B0 I MSGMON   EQU    *
            00B0 I MSGABT   EQU    *
00B0  8E FF8F A             LDS    #BOS           S:=BOTTOM OF STACK
00B3                        SUBR   PMSG
00B5  20 8A                 BRA    MONEN1
                 *
                 ************************************************
                 *
                 * SWI HANDLER:
                 *    DETERMINE WHETHER SWI IS MONITOR CALL, RSRSR CALL,
                 *    OR USER SWI (NOT IMPLEMENTED).
                 *
                 ************************************************
                 *
            00B7 I BREAK1   EQU    *              BREAKPOINT ENTRY
00B7  BD 0005 R             JSR    PINIT          CLEAR PROM BURNER .
00BA  86 80                 LDA A  #128           PRETEND TO BE SWI 128
00BC  20 1A                 BRA    SWI40          SAVE REGS
                 *
            00BE I SWIHAN   EQU    *
                 * FIND INDEX BYTE (BYTE AFTER SWI THAT GOT US HERE)
00BE  30                    TSX
00BF  EE 05                 LDX    5,X            X:=RET. ADR.
00C1  A6 00                 LDA A  0,X            A:=INDEX BYTE
00C3  2B 0C                 BMI    SWI30          BREAKPOINT?
                 * IF USER HAS ADDITIONAL (RS)**3 ADDR OF FIRST+2 MUST BE IN FFF4
00C5  80 18                 SUB A  #24            RSRSR CALL?
00C7  2A 03                 BPL    SWI20          NO --
00C9  7E 0000 R             JMP    RSRSR
                 *
                 *  USER SWI
                 *
00CC  FE FFF4 A  SWI20      LDX    USWI
00CF  6E 00                 JMP    0,X            GO DO IT
                 *
                 * MONITOR CALL.  COPY REGS FROM STACK
                 *
00D1  30         SWI30      TSX                   INCREMENT RET. ADDR.
00D2  6C 06                 INC    6,X
00D4  26 02                 BNE    SWI40
00D6  6C 05                 INC    5,X
00D8  CE FFEB A  SWI40      LDX    #CREG          DEST. FOR 1ST REG
                 * BEGIN LOOP
00DB  33         SWI50      PUL B                 GET REG
00DC  E7 00                 STA B  0,X            COPY
00DE  08                    INX                   MOVE TO NEXT REG
00DF  8C FFF2 A             CPX    #CREG+7        END OF LOOP?
00E2  26 F7                 BNE    SWI50
                 * END LOOP
                 *
                 * S NOW CONTAINS ITS VALUE BEFORE SWI
                 * WAS EXECUTED.  SAVE IT.
                 *
```

```
00E4  AF 00                STS    0,X
                  *
                  * A STILL CONTAINS SWI INDEX.  TEST IT
                  *
00E6  81 81                CMP A  #129
00E8  26 04                BNE    PREGS         NOT 129: BREAK
00EA  8D 07                BSR    PR1           129:  SNAPSHOT
00EC  20 1E                BRA    RESTAK        AND RETURN TO USER PROGRAM
                  *
                  ***********************************************************
                  *
                  * PREGS:  PRINT USER REGISTERS
                  *
                  ***********************************************************
                  *
          00EE I  PREGS    EQU    *
00EE  8D 03                BSR    PR1
00F0  7E 0047 I            JMP    MONEND
          00F3 I  PR1      EQU    *             SUBROUTINE TO PRINT REGS
00F3  CE FFE8 A            LDX    #CREG         X POINTS TO 1ST BYTE OF AREA
                  * PRINT 3 1-BYTE REGS
00F6  C6 03                LDA B  #3            SET UP COUNT
                  *
00F8              PR10     SUBR   P2HEX
00FA  8D 0380 I            JSR    PSPACE
00FD  5A                   DEC B
00FE  2E F8                BGT    PR10
                  *
                  * PRINT 3 2-BYTE REGS
0100  C6 03                LDA B  #3            SET UP COUNT
                  *
0102  8D 037C I   PR20     JSR    P4HEXS
0105  5A                   DEC B
0106  2E FA                BGT    PR20
                  *
0108  8D 0304 I            JSR    PCRLF         PRINT CRLF
010B  39                   RTS                  RETURN
                  *
                  ***********************************************************
                  *
                  * RESTORE USER STATUS AND RETURN FROM MONITOR
                  *
                  ********************************************************
                  *
                  * RESTORE USER'S  STATUS
                  *
010C  8E FFF2 A   RESTAK   LDS    SREG          TOP OF USER STACK
010F  CE FFF1 A            LDX    #CREG+6       USER REGS.
                  *BEGIN LOOP
0112  A6 00       RUS10    LDA A  0,X           GET USER REG
0114  36                   PSH A                PUSH INTO USER STACK
0115  09                   DEX                  MOVE TO NEXT REG
0116  8C FFEA A            CPX    #CREG-1       LAST REG ?
0119  26 F7                BNE    RUS10         NO. CONTINUE LOOP
                  *END OF LOOP
011B  3B                   RTI                  RETURN TO USER PROG
                  ***********************************************************
                  *
                  * COMMANDS AND SUBROUTINES:
                  *
                  ***********************************************************
                  *
                  ***********************************************************
                  *
                  * CHEKSM(CKSM)
                  *  VALIDATE CKSM
                  *
                  ***********************************************************
          011C I  CHEKSM   EQU    *
011C  B6 FFE4 A            LDA A  CKSM          SAVE CALC. CKSM
011F  36                   PSH A
0120  8D 029E I            JSR    NEXT2D        A:= NEXT BYTE FROM TAPE
0123  33                   PUL B
0124  53                   COM B                B:=CALC. CKSM
0125  11                   CBA                  B=TAPE CKSM?
0126  26 01                BNE    CS1           NO.
0128  39                   RTS
                  *
0129  30          CS1      TSX                  X:=ADR OFCALC. CKSM
012A  09                   DEX
012B                       SUBR   P2HEX         PRINT CALC. CKSM
012D  8D 0380 I            JSR    PSPACE
0130  CE 0278 I            LDX    #MCSER        PRINT "CKSM ERR"
0133  7E 0080 I            JMP    MSGABT
                  *
                  ***********************************************************
                  *
                  * DM  ADDL,ADDH   COMMAND
                  *
                  ***********************************************************
          0136 I  DM       EQU    *
0136  8D 35                BSR    GETRNG        GET ADD RANGE FROM BUF
                  *                     RETURNS ADDL,ADDH+1
                  *BEGIN OUTER LOOP
0138  CE FFDC A   DM10     LDX    #ADDL
013B  8D 037C I            JSR    P4HEXS        PRINT ADDL, SPACE
                  * BEGIN INNER LOOP
013E  FE FFDC A   DM20     LDX    ADDL
0141                       SUBR   P2HEX         PRINT MEM(X),SPACE,INC X
0143  8D 0380 I            JSR    PSPACE
0146  FF FFDC A            STX    ADDL
0149  BC FFDE A            CPX    ADDH          IF ADDL=ADDH+1, END OF RANGE
014C  27 0C                BEQ    DM50          EXIT OUTER LOOP
014E  B6 FFDD A            LDA A  ADDL+1        IF LSB'S OF ADDL=0, END OF LINE
0151  84 0F                AND A  #$F
0153  26 E9                BNE    DM20          NOT END OF LINE. CONTINUE
                  * END OF INNER LOOP
0155  8D 0304 I            JSR    PCRLF         PRINT CR,LF
0158  20 DE                BRA    DM10          EXIT INNER LOOP
                  * END OF OUTER LOOP
015A  7E 0041 I   DM50     JMP    MONEN1        CR,LF, BACK TO MONITOR
                  *
                  ***********************************************************
                  *
                  ** PUNCH END OF FILE AND 60 NULLS
                  *
                  ***********************************************************
015D  CE 028A I   EOF      LDX    #MPEOF        PUNCH EOF RECORD
0160                       SUBR   PMSG
                  *
                  ***********************************************************
                  *
                  ** PUNCH END OF FILE AND 60 NULLS
                  *
                  ***********************************************************
015D  CE 028A I   EOF      LDX    #MPEOF        PUNCH EOF RECORD
0160                       SUBR   PMSG
                  *
                  * PUNCH 60 NULLS
                  *
0162  C6 3B       NULLS    LDA B  #59           LOAD COUNTER
                  *                        BEGIN LOOP
0164  4F          NULL1    CLR A                LOAD NULL
0165  8D 02DD I            JSR    OUTCH         PRINT ONE NULL
0168  5A                   DEC B                DECREMENT COUNTER
0169  26 F9                BNE    NULL1         DONE?
                  *                        END OF LOOP
016B  20 ED                BRA    DM50          CR,LF,BACK TO MONITOR
                  ***********************************************************
                  *
                  * GETRANGE (ADDL,ADDH,BUFPTR)
                  *    GET ADDRESS RANGE FROM BUF
                  *    ABORT IF INVALID
                  *     SET ADDH:=ADDH+1 TO SIMPLIFY COMPARISONS
                  *    RETURNS ADDL & ADDH+1
                  *    ALTERS ADR,X,A,B
                  *
                  ***********************************************************
          016D I  GETRNG   EQU    *
016D  8D 0288 I            JSR    NXTADR        GET ADDL
0170  FE FFDA A            LDX    ADR
0173  FF FFDC A            STX    ADDL          STORE ADDL
0176  FF FFDE A            STX    ADDH          MAY BE ONLY 1 PARAM
0179  8D 0288 I            JSR    NXTADR        GET ADDH
017C  27 06                BEQ    GETRG3        ONLY 1 PARAM
                  *
017E  FE FFDA A   GETRG1   LDX    ADR
0181  FF FFDE A            STX    ADDH          SAVE ADDH
                  * THE NEXT 5 INSTR TEST ADDH-ADDL
0184  CE FF90 A   GETRG3   LDX    #BASE         REF W.R.T. BASE OF RAM
```

```
0187  A6 4E                LDA A  ADDH-BASE,X   MSBYTE
0189  E6 4F                LDA B  ADDH+1-BASE,X
018B  E0 4D                SUB B  ADDL+1-BASE,X
018D  A2 4C                SBC A  ADDL-BASE,X
018F  24 06                BCC    GETRG4        ADDH.GE.ADDL
0191  CE 0265 I   RNGERR   LDX    #MRNGER       RANGE ERR MSG
0194  7E 0080 I            JMP    MSGABT        PRINT MSG & ABORT
                  *
0197  FE FFDE A   GETRG4   LDX    ADDH          INC ADDH
019A  08                   INX
019B  FF FFDE A            STX    ADDH
019E  39                   RTS
                  *
                  *
                  ***********************************************************
                  *
                  * GO COMMAND
                  *
                  ***********************************************************
                  *
019F  8D 0288 I   GO       JSR    NXTADR        GET PARAM
01A2  27 06                BEQ    G10           NO PARAM. CONTINUE EXECUTION
                  *
01A4  FE FFDA A            LDX    ADR           ADR=PARAM FROM NXTADR
01A7  FF FFF0 A            STX    PREG
                  *
01AA  7E 010C I   G10      JMP    RESTAK        (IN INTERRUPT HANDLER)
                  *
                  *
                  ***********************************************************
                  *
                  * LOAD COMMAND
                  *
                  ***********************************************************
                  *
          01AD I  LOAD     EQU    *
01AD  CE 0000 A            LDX    #0            INITIALIZE RANGE & OFFSET
01B0  FF FFD8 A            STX    OFFSET        TO 0000-FFFF,0000
01B3  FF FFDC A            STX    ADDL
01B6  09          LOOFST   DEX
01B7  FF FFDE A            STX    ADDH
01BA  8D 0288 I            JSR    NXTADR        ANY OPERANDS?
01BD  27 1E                BEQ    LHF2          NO, USE DEFAULT.
01BF  FE FFDA A            LDX    ADR           YES.
01C2  FF FFD8 A            STX    OFFSET        IF ONE, IT'S OFFSET
01C5  8D 0288 I            JSR    NXTADR        ANOTHER?
01C8  27 13                BEQ    LHF2          NO.
01CA  FE FFD8 A            LDX    OFFSET        YES.  FIRST TWO ARE RANGE
01CD  FF FFDC A            STX    ADDL
01D0  CE 0000 A            LDX    #0
01D3  FF FFD8 A            STX    OFFSET
01D6  8D A6                BSR    GETRG1
01D8  FE FFDE A            LDX    ADDH
01DB  20 D9                BRA    LOOFST        GO TRY AGAIN FOR OFFSET
                  * BEGIN OUTER LOOP
01DD  8D 03A5 I   LHF2     JSR    RDRON         TURN ON READER
                  * SHORT LOOP TO SKIP HDR RECORDS
01E0  8D 70       RDPRE    BSR    FINDS         FIND START OF RECORD
                  *                  SETS (ECHO):=0  ON ENTRY
01E2  8D 0400 I            JSR    WAITTY        RETURNS (A):=TTY I/P
01E5  81 30                CMP A  #'0           IGNORE HDR RECORDS
01E7  27 F7                BEQ    RDPRE
                  * END SHORT LOOP
01E9  B7 FFE2 A            STA A  RECTYP        SAVE RECORD TYPE
01EC  7F FFE4 A            CLR    CKSM
01EF  8D 029E I            JSR    NEXT2D        READ BYTE COUNT FROM TAPE
01F2  4A                   DEC A                DEDUCT ADR & CKSM
01F3  4A                   DEC A
01F4  4A                   DEC A
01F5  B7 FFE3 A            STA A  COUNT         SAVE BYTE COUNT
01F8  8D 029E I            JSR    NEXT2D        READ ADR FIELD FROM TAPE
01FB  B7 FFDA A            STA A  ADR           1ST BYTE
01FE  8D 029E I            JSR    NEXT2D
0201  BB FFD9 A            ADD A  OFFSET+1
0204  B7 FFDB A            STA A  ADR+1         2ND BYTE
0207  B6 FFDA A            LDA A  ADR           CARRY TO FIRST BYTE
020A  B9 FFD8 A            ADC A  OFFSET
020D  B7 FFDA A            STA A  ADR
0210  B6 FFE2 A            LDA A  RECTYP        GET RECORD TYPE (0,1,9)
0213  81 31       LHF3     CMP A  #'1           DATA RECORD ?
0215  26 14                BNE    LHF4          NO
                  *
                  * LOAD DATA RECORD
                  *
                  *BEGIN UNTIL LOOP
0217  8D 029E I   LDR10    JSR    NEXT2D        READ 2 HEX DIGITS FROM
                  *                     TAPE. RETURNS IN A
021A  FE FFDA A            LDX    ADR
021D  8D 03AF I            JSR    SETOFF        STORE IN MEM(X), VERIFY
0220  08                   INX
0221  FF FFDA A            STX    ADR
0224  7A FFE3 A            DEC    COUNT         DOES COUNT=0?
0227  2E EE                BGT    LDR10         NO, CONTINUE LOOP
                  *END UNTIL LOOP
0229  20 04                BRA    LHF9
022B  81 39       LHF4     CMP A  #'9           EOF RECORD ?
022D  26 13                BNE    BADTAP        ILLEGAL RECORD TYPE
                  *
022F  8D 011C I   LHF9     JSR    CHEKSM        CHECK CKSM
0232  B6 FFE2 A            LDA A  RECTYP        GET RECORD TYPE
0235  81 39                CMP A  #'9           EOF RECORD ?
0237  26 A4                BNE    LHF2          NO. CONTINUE LOOP
                  *
                  *END OF OUTER LOOP
                  *
0239  8D 039B I            JSR    RDROFF
023C  CE 026F I            LDX    #MEOF         PRINT "EOF"
023F  7E 0080 I            JMP    MSGMON        AND RETURN TO MONITR LOOP
                  *
0242  8D 039B I   BADTAP   JSR    RDROFF
                  *
0245  CE 0281 I            LDX    #MTAPER       PRINT "TAPE ERR"
0248                       SUBR   PMSG
                  *
                  * ACCEPT NO COMMANDS UNTIL USER PRESSES ESC
                  *
024A  7C FFE9 A            INC    ECHO          SET ECHO
                  *
024D  8D 0400 I   BT1      JSR    WAITTY        ESC CAUSES ABORT
0250  20 FB                BRA    BT1
                  *
                  ***********************************************************
                  *
                  * FIND S
0258  81 53                CMP A  #'S           CHAR = S
025A  26 F9                BNE    FS10          NO
                  *END LOOP
025C  39                   RTS
                  *
                  * MESSAGES
                  *
025D      4241    MBADR    CHAR   /BAD ADR/
025F      4420
0261      4144
0263      52
0264      04               BYTE   4
0265      5241    MRNGER   CHAR   /RANGE ERR/
0267      4E47
0269      4520
026B      4552
026D      52
026E      04               BYTE   4
026F      454F    MEOF     CHAR   /EOF/
0271      46
0272      04               BYTE   4
0273      3F3F    MQUES    CHAR   /????/
0275      3F3F
0277      04               BYTE   4
0278      434B    MCSER    CHAR   /CKSM ERR/
027A      534D
027C      2045
027E      5252
0280      04               BYTE   4
0281      5441    MTAPER   CHAR   /TAPE ERR/
0283      5045
0285      2045
0287      5252
0289      04               BYTE   4
028A      5339    MPEOF    CHAR   /S9030000FC/
028C      3033
```

```
028E  3030
0290  3030
0292  4643
0294  04                 BYTE   4
0295  000A      MCRLFS   BYTE   CR,LF,0,0,0,0,'S,'1,4
0297  0000
0299  0000
0298  5331
0290  04
                *
                **************************************************
                *
                * NEXT 2 DIGITS--
                *   READ NEXT 2 CHAR FROM TTY TAPE AND CONVERT
                *   TO HEX NUMBER IN A REG.  UPDATE CKSM.
                *   RETURN UPDATED CKSM IN B REG.
                *
                **************************************************
        029E I  NEXT2D   EQU    *
029E  BD 0400 I          JSR    WAITTY        GET CHAR
02A1  16                 TAB                  SAVE CHAR IN A
02A2  BD 0400 I          JSR    WAITTY
                *
                * SET UP PARAMS FOR CONVERSION ROUTINE.
                * PUSH ASCII CHARS INTO STACK. POINT X AT STACK.
                * SET A=TYPE OF CONVERSION AND B=# OF CHARS TO CONVERT.
                *
02A5  36                 PSH A
02A6  37                 PSH B
02A7  30                 TSX
02A8  C6 02              LDA B  #2
02AA                     SUBR   CONHB         CONVERT FROM ASCII TO BINARY
02AC  24 94              BCC    BADTAP        IF NON-HEX CHAR, ABORT
                *
02AE  17                 TBA                  UPDATE CKSM
02AF  FB FFE4 A          ADD B  CKSM
02B2  F7 FFE4 A          STA B  CKSM
02B5  31                 INS                  RESTORE STACK PTR
02B6  31                 INS
02B7  39                 RTS
                *
                **************************************************
                *
                * NEXT ADR(BUFPTR,ADR)
                *
                *    SET ADR=0 OR NEXT NUMBER STRING STARTING
                *    AT BUFPTR
                *    LEAVES BUFPTR AT CR,DELIMITER,OR FIRST
                *    CHAR BETWEEN G - Z .
                *    LEAVES (A)= LAST CHAR SCANNED.
                *    LEAVES (B)= LS BYTE OF ADR
                *
                *    RETURNS:  CC= Z  FOR  NO PARAMETER
                *              ABORTS IF NON-HEX PARAMETER
                *
                *
                **************************************************
        02B8 I  NXTADR   EQU    *
02B8  7F FFDA A          CLR    ADR           ADR:= 0
02BB  7F FFDB A          CLR    ADR+1
02BE  BD 0385 I          JSR    PXISTS        IS THERE A PARAMETER?
02C1  26 01              BNE    NA1           YES
02C3  39                 RTS                  RETURN W/NO PARAM  CC=Z
                *
                *
                * SET UP PARAMS FOR ASCII TO HEX CONVERSION
                *
02C4  C6 47     NA1      LDA B  #71           MAX. CHARS TO SCAN
02C6                     SUBR   CONHB
02C8  FF FFE0 A          STX    BUFPTR
02CB  B7 FFDA A          STA A  ADR           SAVE RESULT
02CE  F7 FFDB A          STA B  ADR+1
02D1  A6 00              LDA A  0,X           CHECK TERMINATOR
02D3                     SUBR   ALPNUM        IS CHAR ALPHA?
02D5  25 01              BCS    NA3           YES
02D7  39                 RTS
                *
02D8  7E 00AD I NA3      JMP    ABORT         NO
                *
                *
                **************************************************
                *
                * OUTCH = PRINT CHAR IN A
                * OUTCHX = PRINT CHAR AT MEM(X)
                *   IF CHAR = 'CR', FOLLOW WITH LF & 4 NULLS
                *
                **************************************************
                *
02DB  A6 00     OUTCHX   LDA A  0,X           ENTRY 1
                *
        02DD I  OUTCH    EQU    *
                * FIRST CHECK FOR ESC
02DD  37                 PSH B
02DE  F6 FBCE A          LDA B  ACIAS         ACIA INPUT STATUS
02E1  57                 ASR B                C:=RDRF
02E2  24 0A              BCC    OC10          NO INPUT
02E4  F6 FBCF A          LDA B  ACIAD         READ ACIA
02E7  C1 1B              CMP B  #ESC
02E9  26 03              BNE    OC10          NOT ESC
02EB  7E 00AD I          JMP    ABORT
                *
02EE            OC10     SUBR   PUTA          PRINT CHAR
02F0  81 0D              CMP A  #CR
02F2  26 0E              BNE    OC20          NOT CR. RETURN
                *
02F4  86 0A              LDA A  #LF           PRINT LF
02F6                     SUBR   PUTA
02F8  4F                 CLR A                PRINT 4 NULLS
02F9  C6 04              LDA B  #4
                * BEGIN LOOP
02FB            OCLOOP   SUBR   PUTA
02FD  5A                 DEC B
02FE  26 FB              BNE    OCLOOP
                * END LOOP
0300  86 0D              LDA A  #CR           RESTORE A
                *
0302  33        OC20     PUL B
0303  39                 RTS
                *
                **************************************************
                *
                * PRINT CR,LF,NULL
                *
                **************************************************
0304  86 0D     PCRLF    LDA A  #CR
0306  20 D5              BRA    OUTCH         OUTCH PRINTS LF AFTER CR
                *
                *
                **************************************************
                *
                * PUNCH  ADDL,ADDH
                *     PUNCH MEMORY CONTENTS BETWEEN ADDL & ADDH
                *     IN HEX FORMAT
                *
                **************************************************
0308  BD 016D I PUNCH    JSR    GETRNG        READ ADDL & ADDH+1
030B  CE 0000 A          LDX    #0
030E  FF FFD8 A          STX    OFFSET
0311  8D A5              BSR    NXTADR        ANY OFFSET?
0313  27 06              BEQ    PHF15         NO.
0315  FE FFDA A          LDX    ADR           YES.
0318  FF FFD8 A          STX    OFFSET
                *
                * PUNCH DATA RECORDS UNTIL ADDL = ADDH
                *
        031B I  PHF15    EQU    *
                * BEGIN LOOP
                *
                * CALCULATE DATA LENGTH = MIN(30, ADDH+1-ADR)
                *
031B  F6 FFDF A PHF20    LDA B  ADDH+1        B:=ADDH-ADDL
031E  F0 FFDD A          SUB B  ADDL+1
0321  B6 FFDE A          LDA A  ADDH
0324  B2 FFDC A          SBC A  ADDL
0327  26 04              BNE    PUND10        DIFF .GT. 256
0329  C1 1E              CMP B  #30           LS BYTE .GT. 30?
032B  23 02              BLS    PUND20
                *
032D  C6 1E     PUND10   LDA B  #30           DIFF .GT.30
                *
032F  5C        PUND20   INC B                COUNT:=COUNT+3
0330  5C                 INC B                ...INCLUDES ADDR & CKSM
0331  5C                 INC B
0332  F7 FFE3 A          STA B  COUNT
0335  CE 0295 I          LDX    #MCRLFS
0338                     SUBR   PMSG
033A  5F                 CLR B                B HOLDS CKSM
033B  CE FFE3 A          LDX    #COUNT        PUNCH COUNT
033E  8D 34              BSR    PUNBYT
0340  37                 PSH B
0341  FE FFDC A          LDX    ADDL          COMPUTE OFFSET ADDRESS
0344  B6 FFD8 A          LDA A  OFFSET
0347  F6 FFD9 A          LDA B  OFFSET+1
034A                     SUBR   ADDABX
034C  FF FFDA A          STX    ADR           PUNCH FROM ADR
034F  CE FFDA A          LDX    #ADR
0352  33                 PUL B
                *
0353  8D 1F              BSR    PUNBYT        (INCREMENTS X)
0355  8D 1D              BSR    PUNBYT
0357  FE FFDC A          LDX    ADDL          RESTORE X
                *
                * PUNCH BYTES FROM MEMORY UNTIL COUNT IS EXHAUSTED
                *
                * BEGIN LOOP
035A  8D 18     PREC10   BSR    PUNBYT        (CC=0 IF COUNT=0)
035C  2E FC              BGT    PREC10
                * END LOOP
035E  FF FFDC A          STX    ADDL          SAVE X
0361  CE FFE4 A          LDX    #CKSM         PUNCH CKSM
0364  53                 COM B
0365  E7 00              STA B  0,X           CKSM:=B
0367  8D 0B              BSR    PUNBYT
0369  FE FFDC A          LDX    ADDL
036C  BC FFDE A          CPX    ADDH
036F  26 AA              BNE    PHF20
                * END LOOP
0371  7E 0041 I          JMP    MONEN1
                *
                **************************************************
                *
                * PUNBYT (MEM(X),COUNT,CKSM)
                *     PUNCH BYTE AT MEM(X) AND ADJUST COUNT AND CKSM.
                *     CC=Z IF COUNT=0
                *
                **************************************************
0374  EB 00     PUNBYT   ADD B  0,X           CKSM:=CKSM+MEM(X)
0376                     SUBR   P2HEX         PRINT MEM(X) AS 2 CHAR
0378  7A FFE3 A          DEC    COUNT
037B  39                 RTS
                **************************************************
                *
                * P4HEXS:  PRINT 2 BYTES AT X AS 4 HEX CHARS + 2 SPACES
                *
                **************************************************
                *
037C            P4HEXS   SUBR   P4HEX
037E  8D 00              BSR    PSPACE
                *
                **************************************************
                *
                * PSPACE---PRINT 1 BLANK
                *
                **************************************************
0380  86 20     PSPACE   LDA A  #BLANK
0382                     SUBR   PUTA
0384  39                 RTS
                *
                **************************************************
                *
                *
                * PARAM EXISTS(BUFPTR)     (#BUFPTR) = BUFPTR
                *                              (X) = BUFPTR
                * INC BUFPTR UNTIL CHAR = ALPHA OR CR
                * LEAVE A = MEM(BUFPTR)
                * SET Z IF NO PARAMETER EXISTS
                *
                **************************************************
        0385 I  PXISTS   EQU    *             ENTRY FOR (#BUFPTR)=BUFPTR
```

```
0385  FE FFE0 A          LDX    BUFPTR
           0388 I PXISTX  EQU    *            ENTRY FOR (X) = BUFPTR
                  *BEGIN LOOP
0388  A6 00       PX1     LDA A  0,X          IS CHAR  ALPHANUM ?
038A                      SUBR   ALPNUM
038C  25 07               BCS    PX2          YES, EXIT LOOP
038E  81 0D               CMP A  #CR          IS CHAR CR ?
0390  27 03               BEQ    PX2          YES, EXIT LOOP
0392  08                  INX                 MOVE TO NEXT CHAR
0393  20 F3               BRA    PX1
                  *END LOOP
0395  FF FFE0 A   PX2     STX    BUFPTR
0398  81 0D               CMP A  #CR          SET Z IF NO PARAMETER
039A  39                  RTS
                  ***********************************************
                  *
                  * RDR OFF
                  *     TURNS TAPE RDR OFF:
                  *          ACIA RTS O/P HIGH
                  *          ACIA CHAR  $13 (DC3)
                  *
                  ***********************************************
           039B I RDROFF  EQU    *
039B  86 01               LDA A  #$01         RTS HIGH
039D  B7 FBCE A   ROF90   STA A  ACIAC        SET ACIA CONT REG
03A0  86 13               LDA A  #$13         SEND TTY RDR CONT CHAR
03A2                      SUBR   PUTA
03A4  39                  RTS
                  ***********************************************
                  *
                  * RDR ON
                  * · TURNS TAPE READER ON
                  *         ACIA RTS O/P  LOW
                  *         ACIA CHAR  $11 (DC1)
                  *
                  ***********************************************
           03A5 I RDRON   EQU    *
03A5  86 41               LDA A  #$41         RTS LOW
03A7  B7 FBCE A   RON90   STA A  ACIAC        SET ACIA CONT REG
03AA  86 11               LDA A  #$11         SEND TTY RDR CONT CHAR
03AC                      SUBR   PUTA
03AE  39                  RTS
                  *
                  ***********************************************
                  *
                  * SETMEM(X)
                  * SETS MEM(X):=A AND VERIFY
                  *
                  ***********************************************
                  *
           03AF I SETOFF  EQU    *
03AF  36                  PSH A               FIRST CHECK RANGE:
03B0  B6 FFDC A           LDA A  ADDL         LOW LIMIT
03B3  F6 FFDD A           LDA B  ADDL+1
03B6                      SUBR   SUBXAB       16-BIT SUBTRACT
03B8  2B 0A               BMI    SETOUT       TOO LOW
03BA  B6 FFDE A           LDA A  ADDH         HIGH LIMIT
03BD  F6 FFDF A           LDA B  ADDH+1
03C0                      SUBR   SUBXAB
03C2  24 07               BCC    SETPUL       OK
03C4  32          SETOUT  PUL A               OUTSIDE RANGE LIMITS
03C5  86 FF               LDA A  #255         TYPE DELETE (RUBOUT)
03C7                      SUBR   PUTA         TO SIGNAL FACT TO USER
03C9  20 17               BRA    SETM1        OTHERWISE IGNORE STORE REQUEST
03CB  32          SETPUL  PUL A
           03CC I SETMEM  EQU    *
03CC  A7 00               STA A  0,X
03CE  A1 00               CMP A  0,X          VERIFY
03D0  27 10               BEQ    SETM1        ERROR ?
                  * VERIFY ERROR , PRINT ADR
03D2  FF FFDA A           STX    ADR          SET PARAM FOR P4HEX
03D5  CE FFDA A           LDX    #ADR
03D8  8D C1               BSR    RDROFF
03DA  8D A0               BSR    P4HEXS
03DC  CE 025D I   PBADR   LDX    #MBADR       PRINT 'BAD ADR'
03DF  7E 0080 I           JMP    MSGABT       PRINT MSG & ABORT
                  *
03E2  39          SETM1   RTS
                  ***********************************************
                  *
                  * SM  ADR BYTE1,BYTE2,...
                  *
                  ***********************************************
           03E3 I SM      EQU    *
03E3  8D 0288 I           JSR    NXTADR       ADR:= NEXT PARAM
                  *
03E6  FE FFDA A   SMS     LDX    ADR          SAVE ADR IN ADDL
03E9  FF FFDC A           STX    ADDL
                  *
                  * BEGIN WHILE LOOP
03EC  BD 0288 I   SM10    JSR    NXTADR       ADR:= NEXT PARAM
03EF  27 0C               BEQ    SM30         END OF LINE. EXIT LOOP.
03F1  FE FFDC A           LDX    ADDL         X:= ADD TO BE SET
03F4  17                  TBA                 A:=LS BYTE
03F5  8D D5               BSR    SETMEM       MEM(X):=A, VERIFY
03F7  08                  INX                 MOVE TO NEXT ADD
03F8  FF FFDC A           STX    ADDL
03FB  20 EF               BRA    SM10
                  * END OF LOOP
                  *
03FD  7E 0047 I   SM30    JMP    MONEND
                  ***********************************************
                  *
                  * WAIT FOR TTY(CHAR,ECHO)  (#ECHO)=ECHO
                  *      RETURN NEXT TTY CHAR IN  A
                  *      IF (#ECHO) NOT 0 , ECHO CHAR
                  *
                  ***********************************************
           0400 I WAITTY  EQU    *
                  *LOOP UNTIL INPUT .NE. RUBOUT
0400              W10     SUBR   GETA         READ TTY
0402  81 1B               CMP A  #ESC         ESCAPE ?
0404  26 03               BNE    W20          NO
0406  7E 00AD I           JMP    ABORT        YES, ABORT
0409  81 7F       W20     CMP A  #RUBOUT      RUBOUT ?
040B  27 F3               BEQ    W10          YES CONTINUE LOOP
                  *END UNTIL LOOP
040D  7D FFE9 A           TST    ECHO
0410  27 03               BEQ    W30          NO ECHO
0412  BD 020D I           JSR    OUTCH        ECHO A
0415  39          W30     RTS
                          END
```

```
 SYMBOL TABLE:

ABORT  00AD I   ACIAA  0005 I   ACIAC  FBCE A   ACIAD  FBCF A
ACIAI  FFF6 A   ACIAS  FBCE A   ADDABX 0008 A   ADDH   FFDE A
ADDL   FFDC A   ADR    FFDA A   ALPNUM 0013 A   AREG   FFED A
BADINP 00AD I*  BADTAP 0242 I   BASE   FF90 A   BLANK  0020 A
BOS    FF8F A   BREAK  0002 I   BREAK1 0087 I   BREG   FFEC A
BT1    024D I   BUF    FF90 A   BUFPTR FFE0 A   BURN   0001 R
CHEKSM 011C I   CKSM   FFE4 A   CONHB  0015 A   COUNT  FFE3 A
CR     000D A   CREG   FFEB A   CS1    0129 I   CTABLE 008C I
CTEND  00AD I   DL10   0082 I   DLOOP  007A I   DM     0136 I
DM10   0138 I   DM20   013E I   DM50   015A I   ECHO   FFE9 A
EOF    015D I   EOT    0004 A*  ESC    001B A   FINDS  0252 I
FS10   0255 I   G10    01AA I   GETA   0014 A   GETRG1 017E I
GETRG3 0184 I   GETRG4 0197 I   GETRNG 016D I   GO     019F I
IRQVEC FFF8 A   LAST   FFFF A*  LDR10  0217 I   LF     000A A
LHF2   01DD I   LHF3   0213 I*  LHF4   0228 I   LHF9   022F I
LOAD   01AD I   LOOFST 0186 I   MBADR  025D I   MCRLFS 0295 I
MCSER  0278 I   MEOF   026F I   MONEN1 0041 I   MONEND 0047 I
MONENT 0041 I   MONITR 0047 I   MOVE   0002 R   MPEOF  028A I
MQUES  0273 I   MRNGER 0265 I   MSGABT 0080 I   MSGMON 0080 I
MTAPER 0281 I   NA1    02C4 I   NA3    02D8 I   NEXT2D 029E I
NMIVEC FFFC A   NULL1  0164 I   NULLS  0162 I*  NXTADR 0288 I
OC10   02EE I   OC20   0302 I   OCLOOP 02FB I   OFFSET FFD8 A
OUTCH  020D I   OUTCHX 020B I*  P2HEX  000F A   P4HEX  0010 A
P4HEXS 037C I   PBADR  03DC I   PCRLF  0304 I   PHF15  031B I
PHF20  0318 I   PINIT  0005 R   PMSG   0012 A   PR1    00F3 I
PR10   00F8 I   PR20   0102 I   PREC10 035A I   PREG   FFF0 A
PREGS  00EE I   PROMAD FFD8 A   PRTXD  0009 A*  PSPACE 0380 I
PUNBYT 0374 I   PUNCH  0306 I   PUND10 032D I   PUND20 032F I
PUTA   0011 A   PX1    0388 I   PX2    0395 I   PXISTS 0385 I
PXISTX 0388 I*  RDPRE  01E0 I   RDROFF 039B I   RDRON  03A5 I
READ   0003 R   RECTYP FFE2 A   RESTAK 010C I   RNGERR 0191 I
ROF90  039D I*  RON90  03A7 I*  RSRSR  0000 R   RT10   005F I
RT20   0066 I   RT30   0069 I*  RT90   006C I*  RUBOUT 007F A
RUS10  0112 I   SAVESP FFE5 A*  SAVEX  FFE7 A*  SETM1  03E2 I
SETMEM 03CC I   SETOFF 03AF I   SETOUT 03C4 I   SETPUL 03CB I
SM     03E3 I   SM10   03EC I   SM30   03FD I   SMS    03E6 I*
SREG   FFF2 A   START  0000 I*  START1 0007 I   SUBXAB 0008 A
SWI20  00CC I   SWI30  00D1 I   SWI40  00D8 I   SWI50  00DB I
SWIHAN 00BE I   SWIVEC FFFA A   TCOUNT FFEA A*  USWI   FFF4 A
VFY    0004 R   W10    0400 I   W20    0409 I   W30    0415 I
WAITTY 0400 I   XREG   FFEE A

  CHECKSUM = 075E

  LENGTH OF DSECT =     0 (0000)
  LENGTH OF ISECT =  1046 (0416)

               NO ERRORS,  NO WARNINGS, THIS ASSEMBLY
```

```
PAGE  1  PROM    01/09/76  9:26  PROM BURNER ADDITION TO PHOTO

STMT   LOC   OBJECT  M  SOURCE STATEMENT

                        ********************************************************
                        *                                                      *
                                TITLE  PROM BURNER ADDITION TO PHOTO
                        *                                                      *
                        * PROM BURNER
                        *
                        * VERSION 2.0  01/08/76
                        *
                        * COPYRIGHT 1976 BY AMERICAN MICROSYSTEMS INC.
                        *
                        *
                        ********************************************************
                        *
                        *  ASSEMBLY OPTIONS
                        *
               0001 A   MOVER     EQU    1              0= MOVE ROUTINE EXCLUDED
               000A A   DELAY     EQU    10             POST PROGRAM DELAY, BEFORE VFY  (MS)
                                  OPT    LSKP,LMAC
       0000                       ISEL
       0416                       ORG    $416
                                  REF    MONENT,GETRNG,NXTADR,PXISTS,RNGERR,PBADR
                                  REF    PCRLF,PSPACE,SETMEM,ABORT,MONITR
                                  REF    PROMAD,ADR,ADDL,ADDH,COUNT
                                  DEF    BURN,MOVE,READ,VFY,PINIT
                        *
                        *  PIA LOCATIONS:
                        *
               FBC0 A   PIA       EQU    H'FBC0
               0001 A   V50       EQU    H'FBC1-PIA
               0004 A   PROM      EQU    H'FBC4-PIA
                        *
                        *  STANDARD RAM BUFFER (DEFAULT)
                        *
               FC00 A   RAM       EQU    H'FC00
                        *
                        *  CHARACTER TYPING MACRO
                        *
                        TYPE      MACRO  CHAR
                                 IF CHAR 0
                                 LDAA #CHAR
                                 IEND
                                 CALL PRINTA
                                  MEND
                        *
                        *  RSRSR CALL MACRO
                        *
                        CALL      MACRO  ITEM
                                 SWI
                                 BYTE ITEM
                                  MEND
                        *
                        *  RSRSR CALL LOCATIONS
                        *
               0011 A   PRINTA    EQU    17
               0010 A   P4HEX     EQU    16
                        *
                        *  INITIALIZE PROM BURNER PIA'S
                        *
       0416  CE FBC0 A  PINIT     LDX    #PIA
       0419  86 38                LDAA   #B'00111000    TURN OFF 50V
       041B  AA 01                ORAA   V50,X
       041D  A7 01                STAA   V50,X
       041F  86 3A                LDAA   #B'00111010
       0421  A7 05                STAA   PROM+1,X       R/W TO READ
       0423  A7 07                STAA   PROM+3,X       (HOPE NO DOUBLE-DRIVE HERE)
       0425  6F 06                CLR    PROM+2,X       PROM DATA SET TO INPUTS
       0427  6F 04                CLR    PROM,X
       0429  63 04                COM    PROM,X         SELECT ADDRESS AS OUTPUTS
       042B  86 3E                LDAA   #B'00111110
       042D  A7 05                STAA   PROM+1,X       POINT TO ADDRESS OUTPUT REG.
       042F  39                   RTS
                        *
                        *  TYPE A IN BINARY, ENCLOSED BY SPACES
                        *
       0430  37         P8BIN     PSHB                  SAVE B
       0431  36                   PSHA
       0432  8D 0F                BSR    PSP            PRINT LEADING SPACE
       0434  32                   PULA
       0435  C6 08                LDAB   #8             8 DIGIT COUNTER
       0437  49         :8        ROLA
       0438  36                   PSHA
       0439  86 18                LDAA   #24            (=1/2 ASCII "0")
       043B  49                   ROLA
       043C                       TYPE
                  0   +           IF     0
                      +          LDAA #
                      +           IEND
       043C           +           CALL   PRINTA
       043C  3F       +           SWI
       043D     11    +           BYTE   PRINTA
       043E  32                   PULA
       043F  5A                   DECB
       0440  26 F5                BNE    :8
       0442  33                   PULB
       0443  7E 0007 R  PSP       JMP    PSPACE         PRINT ONE MORE SPACE
                        *
                        *  RAM/PROM ADDRESS SETUP & VALIDATION
                        *
       0446  CE FC00 A  RASV      LDX    #RAM           INTITALIZE POINTERS TO DEFAULT RAM
       0449  FF 000D R            STX    ADDL
       044C  CE FE00 A            LDX    #RAM+512
       044F  FF 000E R            STX    ADDH
       0452  7F 000F R            CLR    COUNT          SET FULL PROM FLAG
       0455  BD 0003 R            JSR    PXISTS         ...IF NO ADDRESS.
       0458  27 06                BEQ    :A1
       045A  BD 0001 R            JSR    GETRNG
       045D  7C 000F R            INC    COUNT
       0460  FE 000D R  :A1       LDX    ADDL           DEFAULT PROM ADDRESS
       0463  FF 0008 R            STX    PROMAD         IS SAME AS START
       0466  BD 0002 R            JSR    NXTADR         TRY FOR PROM ADDRESS
       0469  27 06                BEQ    :A3            NO.
       046B  FE 000C R            LDX    ADR            YES.
       046E  FF 0008 R            STX    PROMAD
       0471  CE 0008 R  :A3       LDX    #PROMAD        VERIFY THAT RANGE <= 512
       0474  0D                   SEC                   (FORCE BORROW)
       0475  E6 07                LDAB   7,X            =ADDH+1
       0477  E2 05                SBCB   5,X            =ADDL+1
       0479  A6 06                LDAA   6,X
       047B  A2 04                SBCA   4,X
       047D  81 02                CMPA   #2             SHOULD BE 1 OR 0
       047F  2C 08                BGE    :A4            TOO BIG.
       0481  EB 01                ADDB   1,X            ALSO SHOULD NOT OVERSTEP PROM
       0483  A9 00                ADCA   0,X
       0485  A8 00                EORA   0,X
       0487  84 FE                ANDA   #H'FE
       0489  26 01                BNE    :A4            IT DOES.
       048B  39                   RTS
       048C  7E 0004 R  :A4       JMP    RNGERR         ADDRESS RANGE ERROR
                        *
                        *  TYPE RAM & ROM ADDRESS & DATA
```

```
                         *
048F  CE 000D R  VERR      LDX    #ADDL        TYPE RAM ADDRESS
0492                       CALL   P4HEX
0492  3F               +   SWI
0493     10            +   BYTE   P4HEX
0494  FE 000D R            LDX    ADDL         NOW THE BYTE THERE
0497  A6 00                LDAA   0,X
0499  8D 95                BSR    P8BIN
049B  B6 FBC6 A            LDAA   PROM+2+PIA   THEN PROM DATA
049E  8D 90                BSR    P8BIN
04A0  CE 0008 R            LDX    #PROMAD      NOW IF ADDRESS (LOW 8)
04A3  A6 01                LDAA   1,X          DOES NOT MATCH RAM ADDRESS,
04A5  A1 05                CMPA   5,X          =ADDL
04A7  27 02                BEQ    :T
04A9                       CALL   P4HEX        PRINT PROM ADDRESS
04A9  3F               +   SWI
04AA     10            +   BYTE   P4HEX
04AB  BD 0006 R  :T        JSR    PCRLF
04AE  86 40                LDAA   #64
04B0  48                   ASLA                EXIT C=0, Z=0, V=1
04B1  39                   RTS
                         *
                         *  PROM ADDRESS SETUP & DATA READ
                         *
04B2  CE 0008 R  ADDRS     LDX    #PROMAD
04B5  A6 01                LDAA   1,X          LOW 8 BITS
04B7  B7 FBC4 A            STAA   PROM+PIA
04BA  A6 00                LDAA   0,X          HIGH BIT
04BC  CE FBC0 A            LDX    #PIA
04BF  48                   ASLA                POSITION IT
04C0  4C                   INCA                WITH DATA REGISTER SELECT
04C1  48                   ASLA
04C2  48                   ASLA
04C3  A8 07                EORA   PROM+3,X     INSERT INTO CONTROL
04C5  84 0C                ANDA   #12
04C7  A8 07                EORA   PROM+3,X
04C9  A7 07                STAA   PROM+3,X
04CB  A6 06                LDAA   PROM+2,X     READ DATA
04CD  39                   RTS
                         *
                         *  PROM VERIFY
                         *
04CE  BD 0446 I  VFY       JSR    RASV         GO SETUP ADDRESSES
04D1  8D 2D      :V        BSR    VFY1         VERIFY ONE LOCATION
04D3  24 02                BCC    :N           NO ERROR, OR PRINTED.
04D5  8D 3A                BSR    JVER         PRINT FIXABLE ERROR.
04D7  8D 11      :N        BSR    INCAD        INCREMENT ADDRESSES
04D9  20 F6                BRA    :V
                         *
                         *  PROM READ
                         *
04DB  BD 0446 I  READ      JSR    RASV         SET UP POINTERS
04DE  8D D2      :R        BSR    ADDRS        READ ONE BYTE
04E0  FE 000D R            LDX    ADDL
04E3  BD 0008 R            JSR    SETMEM       STORE IN RAM
04E6  8D 02                BSR    INCAD        NEXT!
04E8  20 F4                BRA    :R
                         *
                         *  INCREMENT RAM/PROM ADDRESS POINTERS
                         *
04EA  FE 0008 R  INCAD     LDX    PROMAD
04ED  08                   INX
04EE  FF 0008 R            STX    PROMAD
04F1  FE 000D R  INK       LDX    ADDL
04F4  08                   INX
04F5  FF 000D R            STX    ADDL
04F8  BC 000E R            CPX    ADDH
04FB  26 B5                BNE    ADDRS
04FD  7E 000A R  EXIT      JMP    MONITR       EXIT TO MONITOR
                         *
                         *  PROM DATA VERIFY, ONE BYTE
                         *
0500  8D B0      VFY1      BSR    ADDRS        SET UP & READ A BYTE
0502  FE 000D R            LDX    ADDL         COMPARE TO RAM
0505  A1 00                CMPA   0,X
0507  27 07                BEQ    :X           OK: C=0, Z=1, V=0
0509  43                   COMA                NO, IS IT FIXABLE?
050A  AA 00                ORAA   0,X          I.E. NO RAM=0, PROM=1?
050C  43                   COMA
050D  26 02                BNE    JVER         YES.  C=1, Z=0, V=0
050F  4C                   INCA
0510  39         :X        RTS
0511  7E 048F I  JVER      JMP    VERR         NO,  TYPE ERROR
                         *
                         *  PROM BURNER ROUTINE
                         *
0514  BD 0446 I  BURN      JSR    RASV         SET UP PARAMETERS
0517  7D 000F R            TST    COUNT        IF FULL, UNPARAMETERIZED,
051A  26 1B                BNE    :B
051C  7F 0008 R            CLR    PROMAD       DO BLANK CHECK
051F  8D 91      :C        BSR    ADDRS
0521  26 75                BNE    NOGOOD       AHA -- IT ISN'T
0523  CE 0008 R            LDX    #PROMAD      INCREMENT PROM ADDRESS
0526  6C 01                INC    1,X
0528  26 F5                BNE    :C
052A  6C 00                INC    0,X
052C  A6 00                LDAA   0,X
052E  46                   RORA
052F  25 EE                BCS    :C
0531  6F 00                CLR    0,X
0533  20 02                BRA    :B
0535  8D B3      :I        BSR    INCAD        ADVANCE TO NEXT
0537  C6 03      :B        LDAB   #3           SET TRY COUNTER
0539  8D C5                BSR    VFY1         CHECK THIS LOCATION
053B  29 C0                BVS    EXIT         CAN'T PROGRAM 1 TO 0
053D  FE 000D R  :L        LDX    ADDL         GET DATUM
0540  A6 00                LDAA   0,X
0542  CE FBC0 A            LDX    #PIA
0545  A7 06                STAA   PROM+2,X
0547  A6 05                LDAA   PROM+1,X
0549  84 F7                ANDA   #B'11110111  SET R/W TO W
054B  A7 05                STAA   PROM+1,X
054D  A6 07                LDAA   PROM+3,X     TURN IT AROUND
054F  84 FB                ANDA   #B'11111011  (TO OUTPUTS)
0551  A7 07                STAA   PROM+3,X
0553  6F 06                CLR    PROM+2,X
0555  63 06                COM    PROM+2,X
0557  37                   PSHB
0558  C6 14                LDAB   #20
055A  8D 3F                BSR    MSEC         CLEAR TIMER
055C  8D 3D      :P        BSR    MSEC         WAIT 1 MS BETWEEN PULSES
055E  A6 01                LDAA   V50,X
0560  84 F7                ANDA   #B'11110111  SET HIGH VOLTAGE
0562  A7 01                STAA   V50,X
0564  8D 35                BSR    MSEC         5 MS PULSE DURATION
0566  8D 33                BSR    MSEC
0568  8D 31                BSR    MSEC
056A  8A 38                ORAA   #B'00111000  TURN OFF HIGH VOLTAGE
056C  A7 01                STAA   V50,X
056E  5A                   DECB
056F  26 EB                BNE    :P           20 TIMES.
0571  6F 06                CLR    PROM+2,X     CONVERT OUTPUTS TO INPUTS
0573  A6 05                LDAA   PROM+1,X     TURN OFF WRITE
0575  8A 38                ORAA   #B'00111000
0577  A7 05                STAA   PROM+1,X
         10                IF     DELAY        OMIT IF NO POST PROGRAM DELAY
0579  C6 0A                LDAB   #DELAY
057B  8D 1E      :W        BSR    MSEC         DELAY (B) MS
057D  5A                   DECB
057E  26 FB                BNE    :W
                           IEND
0580  33                   PULB
0581  BD 0500 I            JSR    VFY1         CHECK IT:
0584  29 0F                BVS    :J           BAD BIT SHOWED UP
0586  27 AD                BEQ    :I           GOOD
0588                       TYPE   21           NO, TYPE A NAK
         21            +   IF     21           0
0588  86 15            +   LDAA   #21
                       +   IEND
058A                   +   CALL   PRINTA
058A  3F               +   SWI
058B     11            +   BYTE   PRINTA
058C  CE FBC0 A            LDX    #PIA
058F  5A                   DECB                AND TRY AGAIN
0590  26 AB                BNE    :L
0592  BD 048F I            JSR    VERR
      0595 I     :J        EQU    *            GIVE UP
0595  7E 0005 R  JBAD      JMP    PBADR        PRINT "BAD ADDRESS" AND QUIT
0598  7E 0009 R  NOGOOD    JMP    ABORT
                         *
                         *  ONE MILLISECOND DELAY
                         *
059B  6D 05      MSEC      TST    PROM+1,X     WAIT FOR CA1 TO FLOP
059D  2A FC                BPL    MSEC
059F  A1 04                CMPA   PROM,X       CLEAR IT (WITH A DATA READ!)
05A1  39                   RTS
                         *
                         *  MEMORY MOVE
                         *
          1                IF     MOVER
05A2  BD 0001 R  MOVE      JSR    GETRNG       GET SOURCE ADDRESS RANGE
05A5  BD 0002 R            JSR    NXTADR       GET DESTINATION STARTING ADDRESS
05A8  27 EB                BEQ    JBAD         ERROR IF NONE
05AA  FE 000D R  :M        LDX    ADDL         GET BYTE
05AD  A6 00                LDAA   0,X
05AF  FE 000C R            LDX    ADR          STORE IT WITH VERIFY
05B2  BD 0008 R            JSR    SETMEM
05B5  08                   INX                 INCREMENT POINTERS
05B6  FF 000C R            STX    ADR
05B9  BD 04F1 I            JSR    INK          COMPARE TO END
05BC  20 EC                BRA    :M           MORE
                           ELSE
                 MOVE      EQU RAM
                           IEND
                         *
                         *  END OF MODULE
                         *
                           END

SYMBOL TABLE:

ABORT  0009 R   ADDH   000E R   ADDL   000D R   ADDRS  04B2 I
ADR    000C R   BURN   0514 I   COUNT  000F R   DELAY  000A A
EXIT   04FD I   GETRNG 0001 R   INCAD  04EA I   INK    04F1 I
JBAD   0595 I   JVER   0511 I   MONENT 0000 R*  MONITR 000A R
MOVE   05A2 I   MOVER  0001 A   MSEC   059B I   NOGOOD 0598 I
NXTADR 0002 R   P4HEX  0010 A   P8BIN  0430 I   PBADR  0005 R
PCRLF  0006 R   PIA    FBC0 A   PINIT  0416 I   PRINTA 0011 A
PROM   0004 A   PROMAD 0008 R   PSP    0443 I   PSPACE 0007 R
PXISTS 0003 R   RAM    FC00 A   RASV   0446 I   READ   04DB I
RNGERR 0004 R   SETMEM 0008 R   V50    0001 A   VERR   048F I
VFY    04CE I   VFY1   0500 I

CHECKSUM = 93E6

LENGTH OF DSECT =     0 (0000)
LENGTH OF ISECT =   424 (01A8)
               NO ERRORS,  NO WARNINGS, THIS ASSEMBLY
```

AMI's Re-entrant self-relative Subroutine ROM: (RSRSRS) = $(RS)^3$

Edited by Robert A. Stevens

FOREWORD

This software article is the last of a four-part series on the EVK 6800 microcomputer hardware, firmware and supporting software. This month's article covers the EVK re-entrant self-relative subroutine program library software resident in ROM.

INTRODUCTION

The cost of microprocessor software development involves many small items: the cost of assembly time, storage time, transmission time, loading time, design, development, documentation and debug. The cost of many of these items continues to accumulate even though a subroutine library exists for common functions, in particular the time and cost of transmission, loading and ROM pattern generation.

The purpose of Re-entrant Self-Relative Subroutine ROMs $(RS)^3$ is to give the user a hardware subroutine package which exists in the breadboard design from the beginning. The programs are documented, debugged and constitute some of the most commonly performed subroutines that assembly language programmers generate.

CONCEPTS

The $(RS)^3$ uses a number of concepts to allow flexibility in the user environment. The first concept is self-relative programming. This simply means that the program will function correctly regardless of where it is located in memory. The user will need to know where it is located so he can reference it. However, this actual location will only have to be recorded once. The self-relative program uses relative address instructions for program control and the index and stack pointer instructions for data manipulation.

The stack is used for temporary storage of data to prevent $(RS)^3$ from being tied to fixed addresses. This allows the program to be re-entrant; i.e. the program can be called at different times without completing the previous call. This means that the same routine can be called by the interrupt processor as well as by the program which was interrupted. The concept of re-entrant code is not to be confused with recursive code; even through recursive coding could have been used in the subroutine package, it is not.

The subroutine calling mechanism uses the SWI instruction followed by a single byte index for the particular subroutine invoked. This was chosen because the SWI from an internal programming viewpoint is the most convenient and the safest. It is safe because an error in a ROM can be corrected by replacing the subroutine ROM without altering any other user ROM. If direct addresses to subroutine code exist in the user's domain, his ROMs would change if the location of the routine in the $(RS)^3$ changed.

IMPLEMENTATION

The user places the base address of the $(RS)^3$ into the SWI vector address. Each SWI instruction requires an index byte to follow the SWI instruction where the index indicates the function to be executed. After the function is performed, the user program will continue with the instruction following the index byte. In essence, a whole new set of instructions have been created for the user which are two bytes long.

To make the entry easier, a macro call can be provided which will assemble the correct index byte when the function name is used. A set of EQU assembler commands associates the name and the index byte value.

Example:

```
                .
                .
                .
MUL8    EQU         10
MUL16   EQU         11
 DIV8   EQU         12
 DIV16  EQU         13
                .
                .
                .
FUN   MACRO         INDEX
        SWI
        BYTE        INDEX
        MEND
                .
                .
                .
        FUN         MUL8
```

Each (RS)^3 ROM will have the ability to interrogate the index byte and vector to the appropriate subroutine if it is included in the ROM. If the index extends the number of subroutines included on the ROM, the number is subtracted from the temporary index value and the next (RS)^3 ROM is automatically branched to. This allows the user to select any of several subroutine sets, where each set of subroutines is represented by a separate ROM. The selected ROMs are concatenated together into a contiguous region of the user's memory space, and are automatically linked together by the index value. Thus the actual value of the index byte for any particular subroutine is the sum of the total number of subroutines in the physically previous (RS)^3 ROMs plus the offset in its own ROM. It must be noted that address assignments for (RS)^3 ROMs must be made beginning at 1K boundary addresses.

The 2K X 8 ROM provided with the PROTO prototyping system includes a set of (RS)^3 subroutines with a slightly different linkage from the standard (RS)^3 form, although the calling sequence is the same. In particular, the provision for additional subroutines in the form of other (RS)^3 ROMs is limited to a total of 127 subroutines. The first additional (RS)^3 ROM address must be placed in RAM location FFF4 (which can be set via the Set Memory command or modified by an initialization code in a user program). Also, since it is incorporated into a larger program, the whole of which nearly fills the 2K bytes of the ROM, the (RS)^3 part of the ROM does not start on an even page boundary, making it awkward for isolated use. However, the 24 subroutines included in this ROM are available to user program calls with the SWI calling sequence, as described.

(RS)^3 SUBROUTINES

The ROM Subroutine Library (RS)^3 operates on a single SWI (3F) command and a second byte of offset giving the S6800 an additional set of two-byte instructions.

Each of the subroutines in the ROM are described, giving the index for the call, a mnemonic subroutine name, and a descriptive title. A brief description of the subroutine operation is also given.

SUBROUTINE
INDEX NUMBER

MNEMONIC
NAME

FUNCTIONAL TITLE
AND DESCRIPTION

00 PUSHALL Push All Registers

Five bytes are pushed onto the stack, containing, respectively, the Condition Codes, the B and A accumulators, and the Index Register. No registers are altered (except the Stack Pointer, which is decremented by 5).

01 POPALL Pop (= Pull) All Registers

Five bytes are pulled from the stack into the Condition Codes, the B and A accumulators, and the Index Register, respectively. The Stack Pointer is incremented by 5.

02 TXAB Transfer Index Register to A and B

The most significant eight bits of the Index Register are copied to the A accumulator, and the least significant eight bits are copied to the B accumulator.

03 TABX Transfer A and B to Index

Accumulator A is copied to the most significant byte position of the Index Register, and accumulator B is copied to the least significant byte position of the Index Register.

04 XABX Exchange A and B with Index

The contents of the Index Register and the two accumulators are exchanged, A with the most significant byte of X, B with the least significant byte.

05 PUSHX Push Index Register

The contents of the Index Register is pushed onto the stack. The Stack Pointer is decremented by two.

06 PULLX Pop (= Pull) Index Register from stack

Two bytes are pulled from the stack into the Index Register, and the Stack Pointer is incremented by two.

07 ADDXAB Add Index to A and B

Add the contents of the Index Register to the two accumulators, as a 16-bit sum, leaving the result in the two accumulators. The most significant byte is assumed to be in accumulator A. The condition codes are set according to the result.

Condition Codes:
H = carry from bit 11 to bit 12 of sum
N = bit 15 of sum
Z = 1 if sum is zero; else = 0

V = 1 if two's complement overflow
C = carry out of bit 15 of sum

08 ADDABX Add A and B to Index Register

Add the contents of the two accumulators to the Index Register, leaving the 16-bit sum in the Index Register. Accumulator A is assumed to be more significant than accumulator B. The condition codes are set according to the result.

Condition Codes:
H = carry from bit 11 to bit 12 of sum
N = bit 15 of sum
Z = 1 if sum is zero, = 0 otherwise
V = 1 if two's complement overflow
C = carry out of bit 15 of sum

09 ADDAX Add A to Index Register

Add the A accumulator to the contents of the Index Register, and return the sum to the Index Register. The Condition Codes are set according to the result.

Condition Codes: (Same as ADDABX)

0A ADDBX Add B to Index Register

Add the contents of the B accumulator to the Index Register, and leave the sum in the Index Register. The Condition Codes are set according to the result.

Condition Codes: (Same as ADDABX)

0B SUBXAB Subtract Index from A, B

Subtract the contents of the Index Register from accumulators A and B as a 16-bit difference. The Condition Codes are set according to the result.

Condition Codes:
H = undefined
N = bit 14 of difference
Z = 1 if result is zero, = 0 otherwise
V = 1 if two's complement overflow
C = borrow into bit 15 of difference

0C SUBABX Subtract A and B from Index Register

Subtract the contents of the A and B accumulators from the Index Register, leaving the difference in the Index. The Condition Codes are set according to the result.

Condition Codes: (Same as SUBXAB)

0D SUBAX Subtract A from Index Register

Subtract the contents of the A accumulator from the contents of the Index Register and return the difference to the Index Register. The Condition Codes are set according to the result.

Condition Codes: (Same as SUBXAB)

0E SUBBX Subtract B from Index Register

Subtract the contents of the B accumulator from the Index Register, leaving the difference in the Index Register. The Condition Codes are set according to the result.

Condition Codes: (Same as SUBXAB)

0F P2HEX Print Byte in Hex

The byte pointed to be the address in the Index Register is converted to hexadecimal notation in ASCII, and output to the ACIA located as follows: Memory locations FFF6—FFF7 contain an address of a pair of bytes (indirect pointer) which in turn contain the address of the ACIA Status register.

```
FFF7      iL
FFF6      iH
 ...
i+1       aL
i         aH
 ...
a+1       ACIA Data
a         ACIA Status
```

Each byte of the output is stored into the ACIA Data Register after bit 1 of the Status Register is true. The Control Register of the ACIA is not altered, and the Data Register is not read by this routine. The Index Register is incremented past the byte which is output.

10 P4HEX Print Address in Hex

The two bytes in memory pointed to by the Index Register are converted to four ASCII digits and output to the ACIA located at the address pointed to by the pointer pointed to by the byte pair at FFF6—FFF7 (see P2HEX). The Index Register is incremented by two.

11 PRINTA Print the Byte in A

The byte in accumulator A is output to the ACIA, the address of whose address is the locations FFF6—FFF7. No registers are altered except the ACIA Data Register.

12 PMSG Print Message String

A message string, the first byte of which is pointed to by the Index Register, is output to the ACIA, the address of whose address is in locations FFF6—FFF7. The string is terminated by an ASCII EXT (= hex 04), and the Index Register is left pointing to that byte on return.

13 VALAN Validate AlphaNumeric

The character pointed to by the Index Register is analyzed, and the Carry flag is set if it is a letter or digit; if it is not a hexadecimal digit, the Overflow flag is set. Other than the condition codes, no registers are altered.

Exit:Condition Codes:
H = undefined
N = undefined
Z = 0
V = 0 if character in range 0—9, A—F; else = 1

C = 1 if character in range 0—9, A—Z; else = 0

14 INPUTA Input ACIA byte to A

One byte is input from the ACIA, the address of whose address is at location FFF6—FFF7, and this byte is returned to accumulator A. The ACIA is not written to, and except for the A accumulator, no registers are changed. (RS)³ samples bit 0 of the status register of the ACIA, and when it goes to one, reads the Data Register. The input byte has bit 7 removed (set to zero).

15 CONHB Convert Hex String to Binary

A string of characters in memory beginning at the address in the Index Register is scanned for valid Hexadecimal digits; when one is found, it and all immediately following hex digits are converted to a binary number, which is left in the A and B accumulators (A is more significant). When this routine is called, the maximum length of the string is in the B accumulator. On exit, the Carry flag is set to one if the conversion resulted in a valid binary number, and the Index Register is left pointing to the next character in the string, or if the string is exhausted before finding any hex digits, to the last character of the string. Max string length in B is (< 128).

Condition Codes:
H = undefined
N = undefined
Z = undefined
V = undefined
C = 1 if valid number; = 0 if not

16 INDEX Multiply A X B and Add to Index

The contents of the A accumulator is multiplied by the contents of the B accumulator, and the product is added to the Index Register. The Condition Codes are set according to the Result.

Condition Codes: (Same as ADDABX)

17 MUL8 Multiply A Times B

Multiply the contents of the A accumulator times the content of the B accumulator, and leave the product in both accumulators as a 16-bit number, with the most significant part in A. This is an unsigned multiply, and if either or both of the factors is negative (two's complement signed) the product will not be a true signed product of the signed factors, as may be seen in this formula:

$(-n)X(m) = (256-n)Xm = 256m + (-nm)$

The condition codes are nonetheless set according to the result.

Condition Codes:
H = undefined
N = bit 15 of product
V = 0
Z = 1 if product is zero; otherwise = D
C = 0

RS³ ASSEMBLY PROGRAM LISTING

```
PAGE  1  RSRSR   01/09/76  9:28  RSRSR -- REENTRANT SELF RELATIVE SUBROUTINE ROM

STMT  LOC   OBJECT  M  SOURCE STATEMENT

                               TITLE  RSRSR -- REENTRANT SELF RELATIVE SUBROUTINE ROM
                       ****************************************************
                       *
                       * (RS)**3 SUBROUTINE ROM FOR USE WITH PROTO
                       *
                       * VERSION 2.0   01/08/76
                       *
                       * COPYRIGHT 1976 BY AMERICAN MICROSYSTEMS INC.
                       *
                       ****************************************************
                0018 A NITEMS   EQU   24            NUMBER OF ROUTINES
                       *
                       *    CALLING   SEQUENCE    LOC
                       *                          X      SWI
                       *                          X+1    INDEX
                       *                          X+2    NEXT INSTRUCTION
                       *
      0000                     ISEC
      05BE                     ORG    $5BE
                               LOCAL
                               DEF    RSRSR
                       *
                       *    ENTRY IS VIA  LOW ORDER ADDRESS OF  ROM
                       *      <<ADDRESS IS PLACED IN SWI VECTOR ADDRESS
                       *
                05BE I RSRSR    EQU   *
                       *
                       *    GET THE INDEX  VALUE
                       *      DOUBLE IT FOR VECTOR ADDRESS INDEX
                       *
      05BE  30                 TSX                  SP INTO X
                       *  RESTORE STATE OF INTERRUPT AT TIME OF CALL
                       *
                       *
      05BF  EE 05              LDX    5,X           X HAS INDEX ADDRESS
                       *
      05C1  4F                 CLR A
      05C2  E6 00              LDA B  0,X           INDEX  INTO  B
      05C4  58                 ASL B                DOUBLE
      05C5  49                 ROL A
                       *
                       *    A,B  HAS    TWO TIMES INDEX
                       *
                       *     VECTOR OF SUBROUTINE  ADDRESSES  IS AT  512  + ROM BASE
                       *
                       *    FROM HERE TO VECTOR  IS  512 - WHERE WE ARE
      05C6  8D 00              BSR    LOCVV
                       *
                       *
                05C8 I LOCVV    EQU   *             STACK HAS WHERE WE ARE
      05C8  30                 TSX
                       *    A,B  WILL HAVE    INDEX *2 + LOCATION(:A)
      05C9  EB 01              ADD B  1,X
      05CB  A9 00              ADC A  0,X
                       *    ADD  VECTOR  OFFSET
                       *
      05CD  CB 24              ADD B  #(LOCV&H'00FF) LOW ORDER  EIGHT BITS
      05CF  89 01              ADC A  #(LOCV/H'100) HIGH ORDER EIGHT BITS
                       *
                       *    A,B  NOW HAS  ADDRESS OF SUBROUTINE ADDRESS
                       *
      05D1  A7 00              STA A  0,X           SAVE   VECTOR  ADDRESS, HIGH
      05D3  E7 01              STA B  1,X           SAVE   VECTOR  ADDRESS, LOW
      05D5  EE 00              LDX    0,X           LOAD   VECTOR  ADDRESS  INTO  X
      05D7  EB 01              ADD B  1,X
      05D9  A9 00              ADC A  0,X
                       *   ADD  IN  OFFSET  CONTAINED IN VECTOR TABLE
      05DB  30                 TSX
      05DC  A7 00              STA A  0,X
      05DE  E7 01              STA B  1,X
      05E0  A6 02              LDA A  2,X
      05E2  06                 TAP                  STORE OLD STATE INTO CC
      05E3  EE 00              LDX    0,X
                       *
      05E5  31                 INS
      05E6  31                 INS                  CORRECT   SP
                       *    JUMP   TO  SUBROUTINE
      05E7  AD 00              JSR    0,X
                       *
                       *    NORMAL EXIT  FROM  SUBROUTINE
                       *         INCREMENT  RETURN  ADDRESS
      05E9  30                 TSX
      05EA  6C 06              INC    6,X
      05EC  26 02              BNE    *+4
      05EE  6C 05              INC    5,X
                       *    EXIT
      05F0  3B                 RTI

                       *
                       *    STACK ELEMENTS  ARE  STACK POINTER +2 SINCE  JSR
                       *
                0002 A UC       EQU   2             CC  RELATIVE TO  SP
                0003 A UB       EQU   3             B   RELATIVE TO  SP
                0004 A UA       EQU   4             A   RELATIVE TO  SP
                0005 A UXH      EQU   5             XH  RELATIVE TO  SP
                0006 A UXL      EQU   6             XL  RELATIVE TO  SP
                0007 A URH      EQU   7             RH  RELATIVE TO  SP
                0008 A URL      EQU   8             RL  RELATIVE TO  SP
                       *
                       *    PUSH ALL  ONTO  STACK  --  REGISTERS  CORRECT  ON  EXIT
                       *
                0000 A SRH      EQU   0             SYSTEM   RETURN H RELATIVE  TO SP
                0001 A SRL      EQU   1             SYSTEM   RETURN L RELATIVE  TO SP

                       *
                       *    PUSH  ALL  REGISTERS  ONTO  STACK--REGISTERS  UNMODIFIED
                       *
                       *    CURRENT  STACK SP   +1   +2   +3   +4   +5   +6   +7   +8   +9
                       *                        SRH  SRL  CC   B    A    XH   XL   URL  URH
                       *    RESULT   STACK   BEFORE  RETURN TO MAIN  EXIT
                       *SRH SRL  CC   B    A    XH   XL   URL  URH  CC   B    A    XH   XL
                       *
                               LOCAL
                05F1 I PUSHALL  EQU   *
                       *
                       *    MAKE SPACE
                       *
      05F1  34                 DES
      05F2  34                 DES
      05F3  34                 DES
      05F4  34                 DES
      05F5  34                 DES
                       *
                       *    MOVE STACK DOWN
                       *
      05F6  C6 09              LDA B  #9            NINE BYTES TO MOVE
      05F8  30                 TSX
      05F9  A6 05       :S     LDA A  5,X           OFFSET OF 5
      05FB  A7 00              STA A  0,X
      05FD  08                 INX
      05FE  5A                 DEC B
      05FF  26 F8              BNE    :S
                       *
                       *    RECOPY "PUSHED" REGISTERS
                       *
      0601  C6 05              LDA B  #5            FIVE BYTES TO MOVE
      0603  30                 TSX
      0604  A6 02       :C     LDA A  UC,X
      0606  A7 09              STA A  UC+7,X        OFFSET BY 7
      0608  08                 INX
      0609  5A                 DEC B
      060A  26 F8              BNE    :C
                       *
                       *    EXIT   TO  MAIN
                       *
      060C  39                 RTS
                       *
                       *    USER  STACK  IS
                       *                            CC   B    A    XH   XL
                       *                       SP
                       *

                       *
                       *    POP  ALL  REGISTERS
                       *
                               LOCAL
                060D I POPALL   EQU   *
      060D  30                 TSX
                       *    CURRENT  STACK
                       *    SRH  SRL  CC   B    A    XH   XL   URH  URL  CC   B    A    XH   XL
                       *    RESULT   STACK
                       *                       SRH  SRL  CC   B    A    XH   XL   URH  URL
```

```
                        *
                        *    RECOPY "PULLED" REGISTERS
                        *
060E  C6 05                    LDA B  #5            FIVE OF THEM
0610  A6 09             :C     LDA A  UC+7,X        OFFSET OF 7
0612  A7 02                    STA A  UC,X
0614  08                       INX
0615  5A                       DEC B
0616  26 F8                    BNE    :C
                        *
                        *    SHIFT EVERYTHING OVER
                        *
0618  C6 09                    LDA B  #9            NINE BYTES
061A  A6 03             :S     LDA A  URL-5,X
061C  A7 08                    STA A  URL,X         OFFSET 5
061E  09                       DEX
061F  5A                       DEC B
0620  26 F8                    BNE    :S
                        *
                        *    FINALLY INCREMENT SP
                        *
0622  31                       INS
0623  31                       INS
0624  31                       INS
0625  31                       INS
0626  31                       INS
0627  39                       RTS
                        *
                        *    TRANSFER     X    TO   A,B
                        *
0628  30                TXAB   TSX
0629  A6 05                    LDA A  UXH,X         X HIGH
062B  E6 06                    LDA B  UXL,X         X LOW
062D  A7 04             STAB   STA A  UA,X          TO   A
062F  E7 03                    STA B  UB,X          TO   B
                        *
0631  39                       RTS
                        *
                        *    TRANSFER   A,B  TO  X
                        *
0632  30                TABX   TSX
0633  A6 04                    LDA A  UA,X          A
0635  A7 05                    STA A  UXH,X         TO   X HIGH
                        *
0637  A6 03                    LDA A  UB,X          B
0639  A7 06                    STA A  UXL,X         TO   X LOW
                        *
063B  39                       RTS
                        *
                        *    EXCHANGE      A,B  AND  X
                        *
            063C I      XABX   EQU    *
063C  30                       TSX
                        *    CURRENT   STACK
                        *              SRH  SRL  C    B    A    XH   XL   URH  URL
                        *    RESULT                             XL   XH   A    B
                        *
063D  A6 05                    LDA A  UXH,X         PICK UP UX
063F  36                       PSH A
0640  E6 06                    LDA B  UXL,X
0642  8D EF                    BSR    TABX+1        THEN GO TRANSFER A,B TO X
0644  32                       PUL A
0645  20 E6                    BRA    STAB          TO STORE IN A,B
                               LOCAL
                        *    PUSH   X
                        *
            0647 I      PUSX   EQU    *
                        *
                        *    GET  SPACE  IN  SPACE
                        *
0647  34                       DES
0648  34                       DES
0649  30                       TSX
                        *    MOVE   STACK   DOWN  TWO
064A  86 09                    LDA A  #9            MOVE  TOTAL OF  9 BYTES
                        *
064C  E6 02             :A     LDA B  2,X
064E  E7 00                    STA B  0,X
0650  08                       INX
0651  4A                       DEC A
0652  26 F6                    BNE    :A
                        *
                        *    STACK  MOVED -- INSERT   X
0654  30                       TSX
0655  A6 05                    LDA A  UXH,X
0657  A7 09                    STA A  UXH+4,X
0659  A6 06                    LDA A  UXL,X
065B  A7 0A                    STA A  UXL+4,X
                        *
065D  39                       RTS
                        *    STACK  ON  RET
                        *         SRH  SRL  C    B    A    XH   XL   URH  URL  XH   XL
                        *    SP
                        *
                        *    PUL  X
                        *
                               LOCAL
            065E I      PULX   EQU    *
                        *    GET  X   FROM  STACK
                        *
065E  30                       TSX
065F  A6 09                    LDA A  UXH+4,X       CURRENT  X ON  STACK
0661  A7 05                    STA A  UXH,X         REG  X
0663  A6 0A                    LDA A  UXL+4,X
0665  A7 06                    STA A  UXL,X
                        *
                        *    NOW  MOVE  UP  TWO
0667  86 09                    LDA A  #9            BYTE  COUNT
            0669 I      :A     EQU    *
0669  E6 08                    LDA B  8,X
066B  E7 0A                    STA B  10,X
066D  09                       DEX
066E  4A                       DEC A
066F  26 F8                    BNE    :A
                        *    UPDATE    SP
0671  31                       INS
0672  31                       INS
                        *
0673  39                       RTS
                               LOCAL
                        *
                        *    ADD  X     TO   A,B
                        *
            0674 I      ADDXAB EQU    *
0674  30                       TSX
0675  8D C6                    BSR    XABX+1        EASY WAY: EXCHANGE AB & X
0677  8D 05                    BSR    ADDABX+1      ADD OTHER WAY
0679  20 C2                    BRA    XABX+1        THEN EXCHANGE BACK
                        *
                        *    ADD  A,B  TO   X
                        *
067B  30                ADDABX TSX
067C  A6 03                    LDA A  UB,X
067E  E6 04                    LDA B  UA,X
                        *
                        *
                        * CODE SHARED BY ADDABX, INDEX
                        *
            0680 I      ADDAB  EQU    *
0680  AB 06                    ADD A  UXL,X         ADD  UXL TO  UB
0682  A7 06                    STA A  UXL,X         STORE   INTO  UXL
                        *
0684  E9 05                    ADC B  UXH,X         ADD  UAH TO  UA
0686  07                STAUXH TPA                  SAVE      STATUS
0687  E7 05                    STA B  UXH,X         STORE    INTO UXH
                        *
0689  6D 06                    TST    UXL,X         TEST  LOW  BYTE FOR ZERO
                        *
                        * CODE SHARED BY ADDABX, MULB, MUL16
                        *
            068B I      TESTZ  EQU    *
068B  27 02                    BEQ    :A            YES  -- HIGH BYTE STATUS IS TRUE RESULT
068D  84 FB                    AND A  #$FB          NO   -- Z BIT CLEARED
068F  A7 02             :A     STA A  UC,X          SAVE  STATUS
                        *
0691  39                       RTS
                        *
                        *    ADD  A     TO   X
                        *
            0692 I      ADDAX  EQU    *
0692  30                       TSX
0693  A6 04                    LDA A  UA,X
            0695 I      ADDZ   EQU    *
0695  C6 00                    LDA B  #0
0697  20 E7                    BRA    ADDAB
                        *    ADD  B     TO   X
            0699 I      ADDBX  EQU    *
0699  30                       TSX
069A  A6 03                    LDA A  UB,X
069C  20 F7                    BRA    ADDZ
                        *
                        *    SUBTRACT  X     FROM     A,B
                        *
069E  30                SUBXAB TSX
069F  8D 9C                    BSR    XABX+1
06A1  8D 03                    BSR    SUBABX+1
06A3  20 98                    BRA    XABX+1
                        *
                        *    SUBTRACT  A,B  FROM     X
                        *
            06A5 I      SUBABX EQU    *
06A5  30                       TSX
06A6  E6 05                    LDA B  UXH,X
06A8  A6 06                    LDA A  UXL,X
                        *
06AA  A0 03                    SUB A  UB,X
06AC  A7 06                    STA A  UXL,X
                        *
06AE  E2 04                    SBC B  UA,X
06B0  20 D4                    BRA    STAUXH

                        *
                        *    SUBTRACT  A     FROM     X
                        *
            06B2 I      SUBAX  EQU    *
06B2  30                       TSX
06B3  E6 04                    LDA B  UA,X
06B5  A6 06             :SUB   LDA A  UXL,X
06B7  10                       SBA                  SUB  A FROM XL
06B8  A7 06                    STA A  UXL,X         STORE       XL
                        *
06BA  E6 05                    LDA B  UXH,X
06BC  C2 00                    SBC B  #0
06BE  20 C6                    BRA    STAUXH
                        *

                        *
                        *    SUB  B    FROM X
                        *
            06C0 I      SUBBX  EQU    *
06C0  30                       TSX
06C1  E6 03                    LDA B  UB,X
06C3  20 F0                    BRA    :SUB
                        *
                        *
                        * INDEX:  X:=X + A*B    (SAVE USERA,B)
                        *
                               LOCAL
            06C5 I      INDEX  EQU    *
06C5  8D 11                    BSR    MPY8          A,B := USERA*USERB
                        *
                        * EXCHANGE A & B TO SHARE CODE W/ ADDABX
                        *
06C7  37                       PSH B
06C8  16                       TAB
06C9  32                       PUL A
06CA  30                       TSX
06CB  20 B3                    BRA    ADDAB
                        *
                        *
                        * MULB:  A,B := A*B
                        *
            06CD I      MULB   EQU    *
06CD  8D 09                    BSR    MPY8
06CF  30                       TSX
06D0  E7 03                    STA B  UB,X          SAVE RESULT
06D2  A7 04                    STA A  UA,X          SET UP N BIT
06D4  07                       TPA
06D5  5D                       TST B
06D6  20 B3             JMPTZ  BRA    TESTZ         UPDATE USER C & RETURN
                        *
                        *
                        * MUL16--16 BIT MULTIPLY
                        *    A,B,X := A,B*X
                        *
                        *    A,B = PARTIAL PRODUCT
                        *    USERX = MULTIPLIER
                        *    USERA,USERB = MULTIPLICAND
                        *
              0                IF     NITEMS-24     (OMIT IF NITEMS < 25)
                        *
                        *
                        * SUBROUTINE MPY8:  AA,B :=USERA*USERB
                        *    A = PARTIAL PRODUCT
                        *    B = MULTIPLIER & LSB'S OF PAR. PROD.
                        *    USERA = MULTIPLICAND
                        *
                               LOCAL
06D8  86 08             MPY8   LDA A  #8            PUSH COUNTER INTO STACK
06DA  36                       PSH A
                        * STACK = COUNT, R, R, R, R, C, B, A, X, X, R, R
06DB  4F                       CLR A
06DC  30                       TSX
06DD  E6 06                    LDA B  UB+3,X        B=MULTIPLIER
06DF  56                       ROR B
                        *
                        * LOOP 8 TIMES:
                        *
06E0  24 02             :LOOP  BCC    :SHIFT        MULTIPLIER IS EVEN
06E2  AB 07                    ADD A  UA+3,X
                        *
06E4  46                :SHIFT ROR A                SHIFT LSB OF A INTO B
06E5  56                       ROR B
06E6  6A 00                    DEC    0,X           CHECK COUNT
06E8  26 F6                    BNE    :LOOP
                        *
                        * END OF LOOP
                        *
06EA  31                       INS                  RESTORE SP
06EB  39                       RTS
                        *
                        *
                        *    RELATIVE ENTRY POINTS TO SUBROUTINES VECTOR
                        *
            0124 A      LOCV   EQU    *-LOCVV
            06EC I      SELECTOR EQU  *             INDEX
06EC        FF05 A             WORD   PUSHALL-*     0
06EE        FF1F A             WORD   POPALL-*      1
06F0        FF38 A             WORD   TXAB-*        2
06F2        FF40 A             WORD   TABX-*        3
06F4        FF48 A             WORD   XABX-*        4
06F6        FF51 A             WORD   PUSX-*        5
06F8        FF66 A             WORD   PULX-*        6
06FA        FF7A A             WORD   ADDXAB-*      7
06FC        FF7F A             WORD   ADDABX-*      8
06FE        FF94 A             WORD   ADDAX-*       9
0700        FF99 A             WORD   ADDBX-*       10
0702        FF9C A             WORD   SUBXAB-*      11
0704        FFA1 A             WORD   SUBABX-*      12
0706        FFAC A             WORD   SUBAX-*       13
0708        FFB8 A             WORD   SUBBX-*       14
070A        0017 A             WORD   P2HEX-*       15
070C        0010 A             WORD   P4HEX-*       16
070E        002F A             WORD   PRINTA-*      17
0710        004F A             WORD   PRESS-*       18
0712        005D A             WORD   VALA-*        19
0714        0056 A             WORD   INPUTA-*      20
0716        0096 A             WORD   CONV-*        21
0718        FFAD A             WORD   INDEX-*       22
071A        FFB3 A             WORD   MULB-*        23
              0                IF     NITEMS-24
                               LOCAL
                        *
                        *  PRINT 2/4 HEX CHARS FROM MEM(UX,UX+1)
                        *    UX  IS INCREMENTED UPON OUTPUT I.E. UX = UX+2.
                        *
            071C I      P4HEX  EQU    *
071C  30                       TSX
071D  EE 05                    LDX    UXH,X         USERS X
071F  8D 06                    BSR    PHEX          PRINT  MEM()
                        *
                        *  PRINT  2 HEX CHARS FROM  MEM(UX)
            0721 I      P2HEX  EQU    *
0721  30                       TSX
0722  EE 05                    LDX    UXH,X         USERS  X
0724  8D 01                    BSR    PHEX          PRINT MEM (X)
0726  39                       RTS
                        *
                        *  PRINT 2 HEX CHARS FROM MEM(X)
                        *
                                 LOCAL
           0727 I       PHEX     EQU     *
0727   A6 00                     LDA A   0,X          GET THE CHAR
0729   8D 29                     BSR     ASCIIR       CONVERT THE RIGHT NIBBLE AND RESULT IN A
072B   36                        PSH A                SAVE IT
072C   A6 00                     LDA A   0,X          GET CHAR AGAIN
072E   8D 20                     BSR     ASCIIL       CONVERT THE LEFT NIBBLE INTO A
0730   8D 0E                     BSR     PUTAX        PRINT A REG CHAR
0732   32                        PUL A                RECOVER SAVED
0733   8D 10                     BSR     PUTA         ...THEN FALL INTO PINCX
                        ******
                        *
                        *   INCREMENT THE USERS X IN THE STACK
                        *****
                                 LOCAL
           0735 I       PINCX    EQU     *
0735   30                        TSX                  SP IS +2 SINCE TWO BSR DOWN IN CALLS
0736   6C 06                     INC     UXL+2,X      INC MEMORY X LOW
0738   26 02                     BNE     :RTS         OVER FLOW MEANS  INC HIGH PART
073A   6C 07                     INC     UXH+2,X      YES  -- INC HIGH
073C   39               :RTS     RTS                  EXIT
                        *
                        *  PRINT THE CHAR IN USERS A
                        *
           073D I       PRINTA   EQU     *
073D   30                        TSX
073E   A6 04                     LDA A   UA,X         GET CHAR
                                 LOCAL
                        *
                        *  PRINT CHAR  IN DESIGNATED REG
                        *  ACIA ADDRESS IN X
                                 OPT     LMAC
                        *
                        PUT      MACRO   BX
                          PSH  BX  SAVE REG
                        :READY 'LDA  BX  0,X  ACIA  STATUS
                          BIT  BX  #02  READY  ?
                          BEQ  :READY  NOT READY
                          PUL  BX  RESTORE  CHAR
                          STA  BX  1,X  PRINT CHAR
                          RTS
                                 MEND
                        *
                        *  PRINT CHAR IN A
                        *
0740   FE FFF6 A        PUTAX    LDX     H'FFF6       GET INDIRECT ADDRESS OF ACIA
0743   EE 00                     LDX     0,X          GET ACTUAL ADDRESS OF ^ACIA INTO X
0745                    PUTA     PUT     A
0745   36               +        PSH A                SAVE REG
0746   A6 00            +:READY  LDA A   0,X          ACIA  STATUS
0748   85 02            +        BIT A   #02          READY  ?
074A   27 FA            +        BEQ     :READY       NOT READY
074C   32               +        PUL A                RESTORE  CHAR
074D   A7 01            +        STA A   1,X          PRINT CHAR
074F   39               +        RTS
                                 LOCAL
                        *
                        *
                        *  CONVERT  A FROM HEX TO ASCII LEFT/RIGHT NIBBLE
                        *    LEFT PART
                        *
           0750 I       ASCIIL   EQU     *
0750   44                        LSR A                A HAS CHAR TO BE CONVERTED
0751   44                        LSR A
0752   44                        LSR A
0753   44                        LSR A
                        *  INPOSITION
0754   84 0F            ASCIIR   AND A   #H'0F        CLEAR LEFT PART
0756   8B 30                     ADD A   #H'30
0758   81 39                     CMP A   #H'39        0 TO 9
075A   23 02                     BLS     :RTS         YES DONE
075C   8B 07                     ADD A   #7           NO  THEN A TO F
075E   39               :RTS     RTS
                        *
                        *  PRINT MESSAGE  POINTED TO BY X AND TERMINATED BY ETX
                        *
                                 LOCAL
           075F I       PMESS    EQU     *
075F   30                        TSX
0760   EE 05                     LDX     UXH,X        GET USERS X
0762   A6 00                     LDA A   0,X          GET  CHAR
0764   81 04                     CMP A   #ETX         IS IT TERMINATOR
0766   27 06                     BEQ     :RTS         DONE
0768   8D D6                     BSR     PUTAX        PRINT A
076A   8D C9                     BSR     PINCX        INC USERS X
076C   20 F1                     BRA     PMESS        LOOP TILL DONE
076E   39               :RTS     RTS
           0004 A       ETX      EQU     H'04
                        *
                        *  X HAS ADDRESS OF  CHAR TO TO BE TESTED
                        *      FOR BEING ALPHA NUMERIC
                        *      CARRY SET IF TRUE
           076F I       VALAN    EQU     *
                                 LOCAL
076F   30                        TSX
0770   EE 05                     LDX     UXH,X        GET CHAR ADDRESS
0772   8D 05                     BSR     ALPNUM       TEST MEM(X)=ALPHANUMERIC
                        *
                        * SET USER'S CARRY = CURRENT CARRY (AND OTHER FLAGS!)
                        *
0774   07               SCARRY   TPA
                        *
0775   30               SETUS    TSX
0776   A7 02                     STA A   UC,X
0778   39                        RTS
                        *
                        *
                        * SET CARRY IF MEM(X) IS ALPHANUMERIC
                        * CLEAR V IF HEX DIGIT
                        *
           0779 I       ALPNUM   EQU     *
0779   A6 00                     LDA A   0,X          GET THE CHAR
077B   81 41                     CMP A   #'A
077D   2D 0E                     BLT     :NUM         TOO SMALL FOR ALPHA ,IS IT NUMERIC
077F   81 5A                     CMP A   #'Z
0781   2E 12                     BGT     :NOTOK
0783   81 47                     CMP A   #'G          SET V IF >F
0785   29 10                     BVS     :RTS         QUIT IF NOT HEX (C=1)
0787   80 07                     SUB A   #7           CONVERT LETTER TO HEX
0789   84 0F            :OK      AND A   #15          STRIP OVERBITS FROM HEX DIGIT
078B   0D                        SEC                  SET C FOR VALID A/N
078C   39                        RTS
                        *
078D   81 30            :NUM     CMP A   #'0          NUMERIC TESTING
078F   2D 04                     BLT     :NOTOK       NOT NUMERIC
0791   81 39                     CMP A   #'9
0793   2F F4                     BLE     :OK          IT IS IN 0-9
0795   0C               :NOTOK   CLC                  RESET CARRY FOR NOT A/N
0796   0B                        SEV                  SET V FOR NOT HEX EITHER
0797   39               :RTS     RTS
0798   20 9B            JPINCX   BRA     PINCX        EXTRA BRA TO REACH PINCX
                        ***************************************************
                        *
                        * INPUTA:
                        *    INPUT ACIA DATA INTO A REG
                        *    STRIP PARITY
                        *
                        ***************************************************
                                 LOCAL
           079A I       INPUTA   EQU     *
079A   FE FFF6 A                 LDX     H'FFF6       GET ACIA INDIRECT ADDRESS
079D   EE 00                     LDX     0,X          GET ACIA ADDRESS
                        *
079F   A6 00            :WAIT    LDA A   0,X          ACIA STATUS
07A1   47                        ASR A                CARRY:=RDRF
07A2   24 FB                     BCC     :WAIT        NO INPUT.  LOOP.
                        *
07A4   A6 01                     LDA A   1,X          ACIA DATA
07A6   84 7F                     AND A   #H'7F        STRIP PARITY
07A8   30                        TSX                  PUT RESULT ONTO STACK
07A9   A7 04                     STA A   UA,X
07AB   39                        RTS
                        *
                        ***************************************************
                        *
                        * CONHB--CONVERT HEX TO BINARY:
                        *    SCAN UP TO B ASCII CHARACTERS STARTING AT X
                        *    LOOKING FOR A VALID HEX NUMBER.  RETURN BINARY
                        *    EQUIVALENT OF NUMBER IN A,B.  IF NUMBER HAS MORE THAN
                        *    16 BITS, IGNORE MSB'S.
                        *
                        * INPUT:    X=ADDRESS OF 1ST CHAR TO BE SCANNED.
                        *           B=MAX. # OF CHARS TO BE SCANNED.
                        *
                        * OUTPUT:   A,B=BINARY RESULT
                        *           CARRY=1 IF VALID NUMBER IS FOUND
                        *           X POINTS TO LAST CHAR SCANNED
                        *
```

```
                        ***************************************************
                                 LOCAL
           07AC I       CONHB    EQU     *
07AC   30                        TSX
07AD   E6 03                     LDA B   UB,X         GET MAX COUNT
07AF   6F 04                     CLR     UA,X         CLEAR USER'S A,B REGS
07B1   6F 03                     CLR     UB,X
                        *
                        * LOOP WHILE NOT ALPHANUMERIC AND COUNT > 1
                        *
07B3   30               :LOOP1   TSX
07B4   EE 05                     LDX     UXH,X        GET CHAR ADDRESS
07B6   8D C1                     BSR     ALPNUM       IS MEM(X) ALPHANUMERIC?
07B8   25 09                     BCS     :FOUND       YES. STOP SCANNING
07BA   5A                        DEC B                DEC COUNT
07BB   2F 04                     BLE     :ENDCNT      COUNT EXHAUSTED
07BD   8D D9                     BSR     JPINCX       INC USER'S X
07BF   20 F2                     BRA     :LOOP1
                        *
                        * END LOOP
                        *
                        * COUNT EXHAUSTED  WITH NO SUCCESS.
                        * (CARRY WAS RESET BY ALPNUM)
                        *
07C1   20 B1            :ENDCNT  BRA     SCARRY       RESET USER C AND RETURN
                        *
                        * WHILE HEX AND COUNT > 0 SHIFT MEM(X) INTO UA,UB
                        *
                        * BEGIN OUTER LOOP
07C3   30               :FOUND   TSX
07C4   EE 05                     LDX     UXH,X
07C6   8D B1                     BSR     ALPNUM       CNVT MEM(X) TO HEX
07C8   29 18                     BVS     :NOGOOD      INVALID CHAR
07CA   37                        PSH B                SAVE COUNT
07CB   C6 04                     LDA B   #4           LOOP COUNT
                        *
                        * SHIFT LEFT UA,UB
                        *
07CD   30                        TSX
07CE   68 04            :SLOOP   ASL     UB+1,X       +1 TO COMP. FOR PUSH
07D0   69 05                     ROL     UA+1,X
07D2   5A                        DEC B
07D3   2E F9                     BGT     :SLOOP
                        *
07D5   AA 04                     ORA A   UB+1,X       'OR' IN NEW CHAR
07D7   A7 04                     STA A   UB+1,X
07D9   33                        PUL B                RETRIEVE COUNT
07DA   8D BC                     BSR     JPINCX       INC USER X
07DC   5A                        DEC B
07DD   2E E4                     BGT     :FOUND       REPEAT
                        * END OUTER LOOP
07DF   0D                        SEC                  VALID NUMBER
07E0   20 92                     BRA     SCARRY       SET USER C AND RETURN
                        *
                        * NON-HEX CHAR FOUND.  IF CHAR = G-Z, THIS IS NOT A VALID
                        * HEX NUMBER.  OTHERWISE, CHAR IS A DELIMITER AND
                        * NUMBER IS VALID.
                        *
07E2   07               :NOGOOD  TPA                  TOGGLE CARRY BIT
07E3   4C                        INC A
07E4   20 8F                     BRA     SETUS        SETUP USER STATUS & RETURN
                        *
  SYMBOL TABLE:                  END

  ADDAB   0680 I   ADDABX 0676 I   ADDAX  0692 I   ADDBX  0699 I
  ADDAAB  0674 I   ADDZ   0695 I   ALPNUM 0779 I   ASCIIL 0750 I
  ASCIIR  0754 I   CONHB  07AC I   ETX    0004 A   INDEX  06C5 I
  INPUTA  079A I   JMPIZ  0606 I*  JPINCX 0798 I   LOCV   0124 A
  LOCVV   05C8 I   MPY8   0608 I   MULB   06CD I   NITEMS 0018 A
  P2HEX   0721 I   P4HEX  071C I   PHEX   0727 I   PINCX  0735 I
  PMESS   075F I   POPALL 0600 I   PRINTA 073D I   PULX   065E I
  PUSHAL  05F1 I   PUSX   0647 I   PUTA   0745 I   PUTAX  0740 I
  RSRSR   05BE I   SCARRY 0774 I   SETUS  0775 I   SRH    0000 A*
  SRL     0001 A*  STAB   0620 I   STAUXH 0686 I   SUBABX 0645 I
  SUBAX   0682 I   SUBBX  06C0 I   SUBXAB 069E I   SVECTO 06EC I*
  TABX    0632 I   TESTZ  0666 I   TXAB   0628 I   UA     0004 A
  UB      0003 A   UC     0002 A   URH    0007 A*  URL    0008 A
  UXH     0005 A   UXL    0006 A   VALAN  076F I   XABX   063C I
CHECKSUM = C70E

LENGTH OF DSECT =      0 (0000)
LENGTH OF ISECT =    552 (0228)

NO ERRORS, NO WARNINGS, THIS ASSEMBLY
```