CPU OPERATION

PolyMorphic
Systems

460 Ward Drive   Santa Barbara   California 93111   (805) 967-2351

This manual is PolyMorphic Systems part number 810112

Copyright 1977, Interactive Products Corporation.

Central Processor Unit Theory of Operation

TABLE OF CONTENTS

1. INTRODUCTION

The 8080 CPU card is a member of PolyMorphic Systems' complete line of microcomputer system components. The CPU card is compatible with the industry standard S-100 bus. In addition to the 8080 CPU chip, the card also contains 512 bytes of read-write memory, sockets for 3072 bytes of read-only memory, an optional serial I/O port, a real-time clock interrupt, an instruction single-step interrupt, and a priority encoder for the 8-level priority interrupt system.

The 8080 CPU contains six 8-bit general purpose registers and an accumulator. The six general purpose registers may be addressed individually or in pairs, providing both single and double precision operators. The 8080 has a 16-bit program counter, allowing direct addressing of up to 64K bytes of memory. A 16-bit stack pointer addresses an external last in/first out stack which may be located in any part of memory. The program counter, flags, accumulator, or general purpose registers may be stored and retreived from this stack. The 8080 instruction set includes arithmetic and logical operations between the accumulator and general-purpose registers, immediate data, or memory. Data may be moved between the accumulator, general purpose registers, and memory in 8 or 16-bit words. Double precision arithmetic and stack manipulation operators extend the capability of the 8080. Conditional and unconditional jump and call instructions are provided, as are computed jumps. Instructions are variable length and execute in 2-18 microseconds.

The CPU card provides 512 bytes of 450 nS RAM and sockets for three 2308 ROMs or 2708 EPROMs. The ROM sockets normally hold the resident portion of the operating system on the POLY 88 and System 88 microcomputer systems. They may be used for the user's own software in custom applications.

A programmable serial communications interface using an 8251 USART is included on the card (optional when CPU card is purchased separately as a kit). The mode of operation (synchronous or asynchronous), data format, control character format, parity, and transmission rate are all under program control. Two sockets on the CPU card allow the connection of minicards providing RS-232 or current loop, and Byte or Polyphase cassette interfaces.

A 50 or 60 Hertz real time clock interrupt is provided on the card along with a single step interrupt. The single step interrupt allows single insruction execution, for debugging purposes, under software control. Eight vectored interrupts are provided on the card for use with these or external interrupts.

2. OPERATIONAL THEORY

2.1 CPU

The 8080 CPU, 8224 clock generator, and 74LS174 status latch perform all system processing operations and provide a timing reference for all other circuitry on the card and the S-100 bus. The CPU generates all of the address and control signals necessary to access memory and I/O ports both on the CPU card and the S-100 bus. The CPU responds to interrupts originating from devices on the bus or the CPU card. The priority encoder forms RST (restart) instructions vectoring the CPU to a different location in response to each interrupt.

The activities of the CPU set are cyclical. The CPU fetches an instruction, performs the operations required, fetches the next instruction, and so on. This orderly sequence of events requires precise timing. The 8224 clock generator provides the primary timing reference for the system. The 8224 produces the two-phase timing inputs (01 and 02) for the 8080. The 01 and 02 signals define a cycle of approximately 540ns duration. All processing activities of the CPU set are referred to the period of 01 and 02 clock signals.

Within the 8080 CPU set, an instruction cycle is defined as the time required to fetch and execute an instruction. During the fetch, a selected instruction (one, two, or three bytes) is extracted from memory and deposited in the CPU's operating registers. During the execution part, the instruction is decoded and translated into specific processing activities.

Every instruction cycle consists of one, two, three, four, or five machine cycles. A machine cycle is required each time the CPU accesses memory or an I/O port. The fetch portion of an instruction cycle requires one machine cycle for each byte to be fetched. The duration of the execution portion of the instruction cycle depends on the kind of instruction that has been fetched. Some instructions do not require any machine cycles other than those necessary to fetch the instruction; other instructions, however, require additional machine cycles to write or read data to or from memory or I/O devices.

Each machine cycle consists of three, four, or five states. A state is the smallest unit of processing activity and is defined as the interval between two successive positive-going transitions of the 01 clock pulse.

There are three exceptions to the defined duration of a state. They are the WAIT state, the hold(HLDA) state and the halt(HLTA) state. Because WAIT, HLDA, and HLTA states depend upon external events, they are by their nature of indeterminate length. Even these exceptional states, however, must be synchronized with the pulses of the driving clock. Thus the duration of all states, including these, are integral multiples of the clock pulse.

To summarize, then, each clock period marks a state; three to five states comprise a machine cycle; and one to five machine cycles comprise an instruction cycle. A full instruction cycle requires anywhere from four to seventeen states for its completion, depending on the kind of instruction involved.

There is just one consideration that determines how many machine cycles are required in any given instruction cycle: the number of times that the processor must reference a memory address or an I/O address, in order to fetch and execute the instruction. Like many processors, the 8080 is so constructed that it transmits one address per machine cycle. Thus, if the fetching and execution of an instruction requires two memory references, then the instruction cycle associated with that instruction consists of two machine cycles. If five such references are called for, then the insruction cycle contains five machine cycles.

Every instruction cycle has at least one reference to memory, during which the instruction is fetched. An instruction cycle must always have a fetch, even if the execution of instruction requires no further references to memory. The first machine cycle in every instruction cycle is therefore a FETCH. Beyond that, there are no fast rules. It depends on the kind of instruction. The input (INP) and the output (OUT) instructions each require three machine cycles: a FETCH, to obtain the instruction; a MEMORY READ, to obtain the address of the object peripheral; and an INPUT or an OUTPUT machine cycle, to complete the transfer.

Every machine cycle within an instruction cycle consists of three to five active states (referred to as T1, T2, T3, T4, and T5). The actual number of states depends upon the instruction being executed, and on the particular machine cycle within the greater instruction cycle. Figure 2 shows the timing relationships in a typical FETCH machine cycle. Events that occur in each state are referred to transitions of the 01 and 02 clock pulses.

At the beginning of each machine cycle (in state T1), the 8080 activates its SYNC output and issues status information onto the status bus just after the rising edge of 01 of state T2. The status information is also available on the data bus during T1.

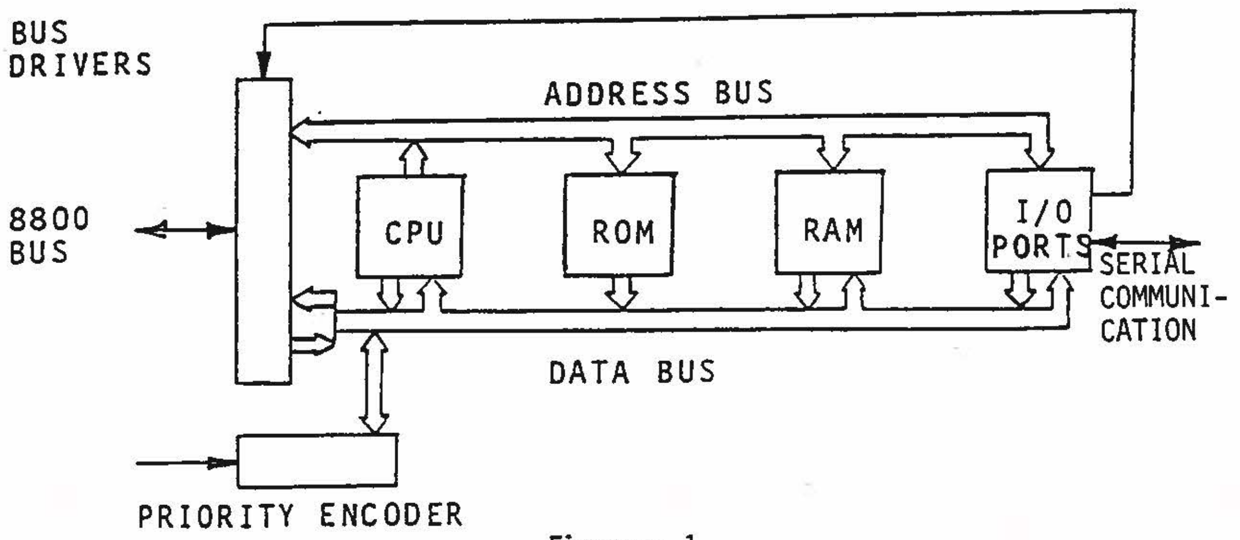


Figure 1.

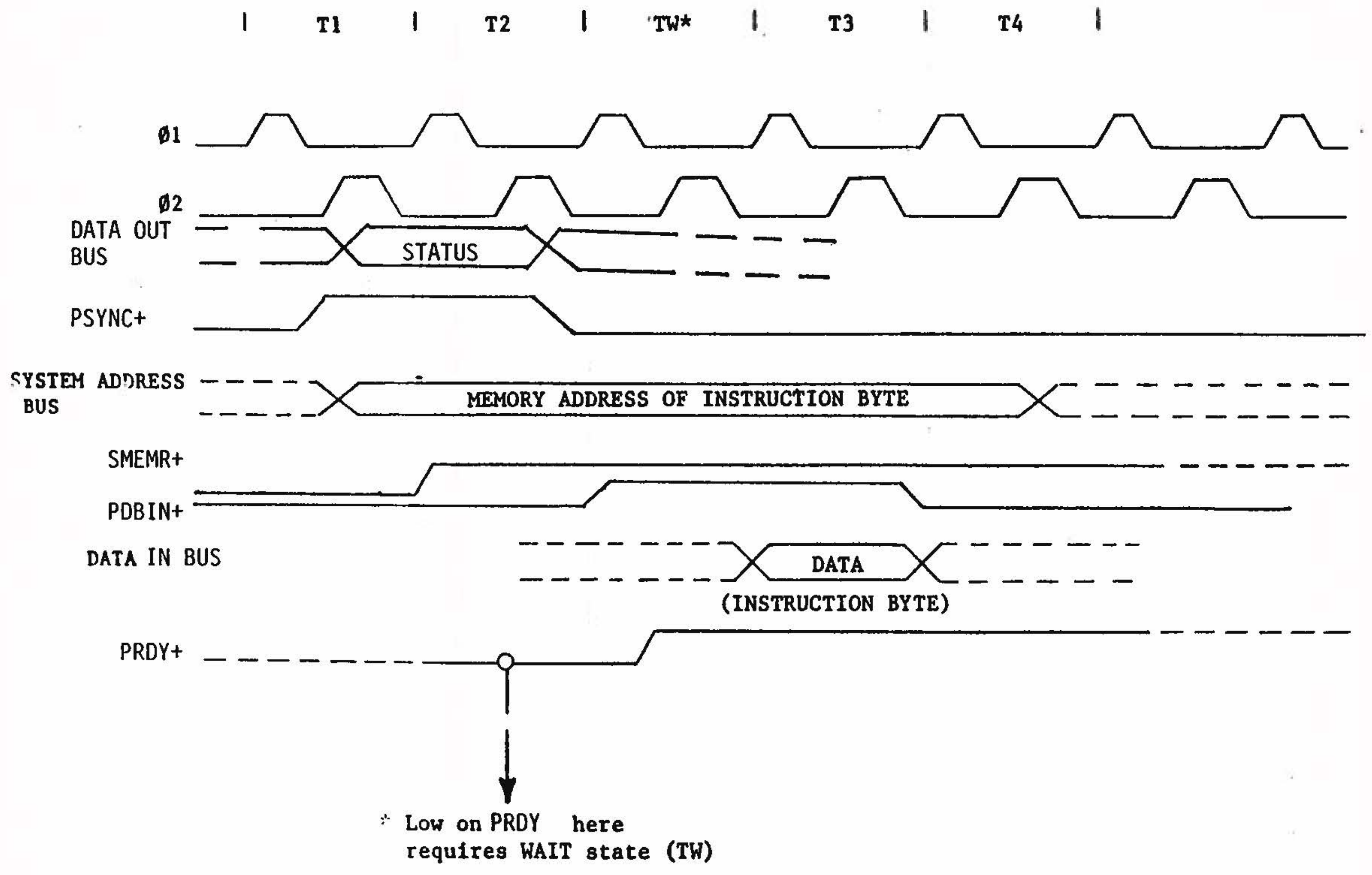


FIGURE 2 . TYPICAL FETCH MACHINE CYCLE

The status information indicates the type of machine cycle in progress. SINTA goes high during an interrupt acknowledgement cycle; SMEMR, when high, signals a memory read cycle; SINP and SOUT, respectively, indicate input and output data transfers. SWO goes low during output or memory write cycles, and HLTA is active (high) only when the CPU is halted. Note that there is not a separate status line for a memory write cycle. This may be generated by the logic SWO -SOUT. After PSYNC returns low, PDBIN goes high, indicating the CPU is expecting data on the Data In Bus.

The rising edge of 02 during T1 loads the address lines A0-A15. These lines become stable within a brief delay of the 02 clocking pulse, and they remain stable until the first 02 pulse after state T3. This gives the processor ample time to read the data returned from memory.

Once the processor has sent an address to memory, there is an opportunity for the memory to request a WAIT. This it does by pulling the PRDY+ line low. As long as the PRDY line remains low, the CPU Set will idle, giviing the memory time to respond to the addressed data request. The processor responds to a wait request by entering an alternative state (TW) at the end of T2, rather than proceeding directly to the T3 state. A wait period may be of indefinite duration. The 8080 remains in the waiting condition until its READY line again goes high. The cycle may then proceed, beginning with the rising edge of the next 01 clock. A WAIT interval will therefore consist of an integral number of TW states and will always be a multiple of the clock period.

The events that take place during the T3 state are determined by the kind of machine cycle in progress. In a FETCH machine cycle, the CPU interprets the data on the Data In Bus as an instruction. During a MEMORY READ, signals on the same bus are interpreted as a data word. The CPU itself outputs data on this bus during a MEMORY WRITE machine cycle. And during I/O operations, the CPU may either transmit or receive data, depending on whether an INPUT or an OUTPUT operation is involved. Consider the following two examples.

Figure 3 illustrates the timing that is characteristic of an input instruction cycle. During the first machine cycle M1), the first byte of the two-byte IN instruction is fetched from memory. The 8080 places the 16-bit memory address on the address bus near the end of state T1. The memory read status signal (SMEMR) during state T2. During the next machine cycle (M2), the second byte of instruction is fetched. During the third machine cycle (M3), the IN instruction is executed. The 8080 duplicates the 8-bit I/O address on address lines A0-7 and A8-15. The 8080 activates the I/O read status signal (SINP) during states T2 of this cycle.

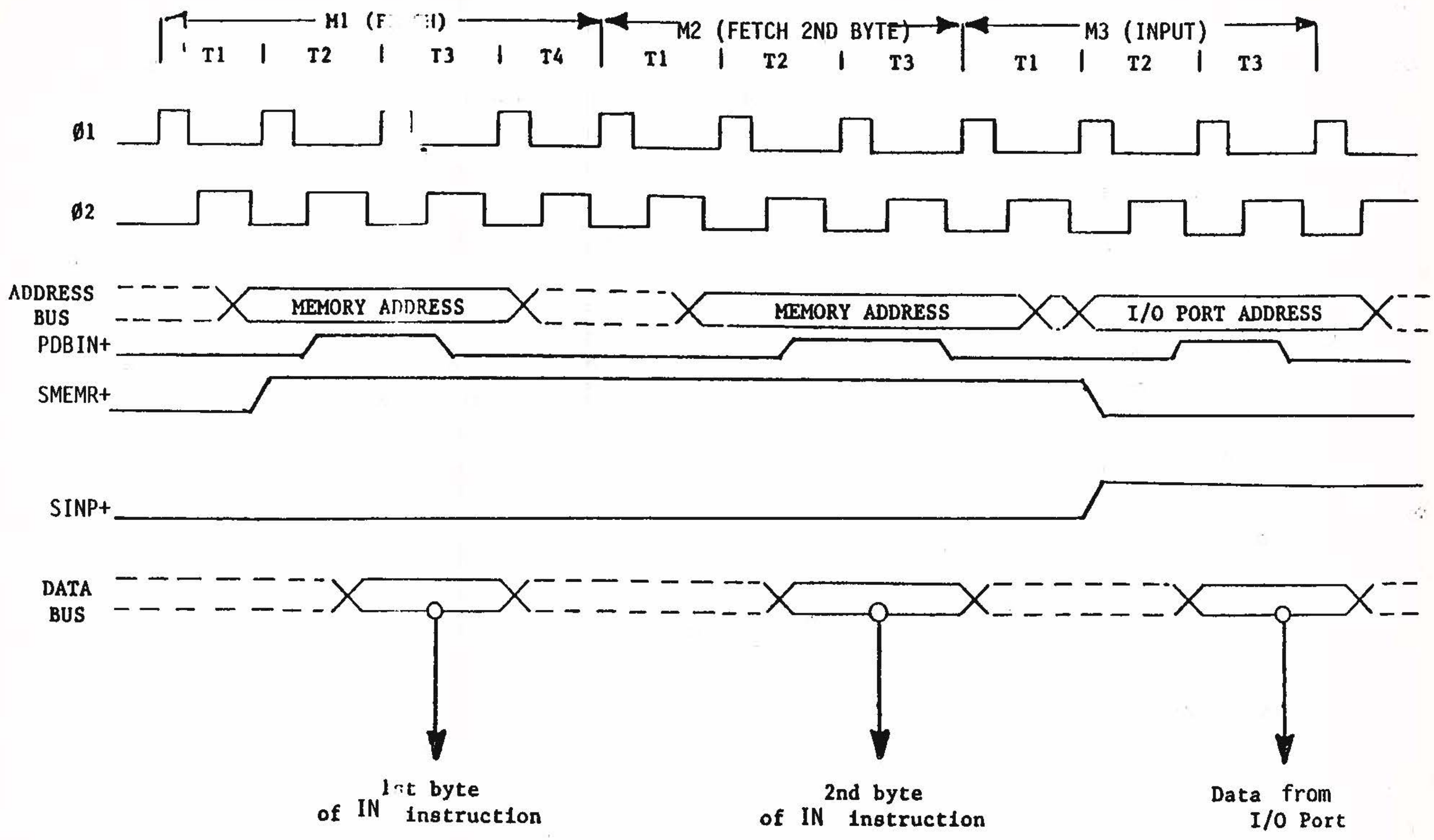


FIGURE 3. INPUT INSTRUCTION CYCLE

Figure 4 illustrates an instruction cycle during which the CPU outputs data. During the first two machine cycles (M1 and M2), the CPU fetches the two-byte OUT instruction . During the third machine cycle (M3), the OUT instruction is executed. The 8080 duplicates the 8-bit I/O address on lines A0-7 and A8-15. The 8080 activates the I/O address on status signal (SOUT) at the beginning of state T2 of this cycle. The 8080 outputs the data onto the Data Out Bus at the end of state T2. Data on the bus remains stable throughout the remainder of the machine cycle.

Observe that a PRDY signal is necessary for completion of an output machine cycle. Unless such an indication is present, the processor enters the TW state, following the T2 state. Data on the output lines remains stable in the interim, and the processing cycle will not proceed until the PRDY line again goes high.

The negative-going leading edge of WR is referred to the rising edge of the first 01 clock pulse following T2. Thus, WR coincides with the appearance of stable data on the system bus. WR remains low until re-triggered by the leading edge of 02, during the state following T3. Note that any TW states intervening between T2 and T3 of the output machine cycle will necessarily extend WR/.

All processor machine cycles consist of at least three states: T1, T2, and T3, as just described. If the CPU has to wait for a PRDY response, then the machine cycle may also contain one or more TW ferred to or from the CPU.

After the T3 state, however, it becomes difficult to generalize. T4 and T5 states are available, if the execution of a particular instruction requires them. But not all machine cycles make use of these states. It depends upon the kind of instruction being executed, and on the particular machine cycle within the instruction cycle. The processor will terminate any machine cycle as soon as its processing activities are completed, rather than proceeding through the T4 and T5 states every time. Thus the 8080 may exit a machine cycle following the T3, the T4, or the T5 state and proceed directly to the T1 state of the next machine cycle.

2.1.1 INTERRUPT SEQUENCES

The 8080 has the built-in capacity to handle external interrupt requests. Peripheral logic can initiate an interrut simply by driving any vectored interrupt line low. The interrupt input is asynchronous, and a request may therefore originate at any time during any instruction cycle. An interrupt request acts in coincidence with the 02 clock to set the internal interrupt latch. This event takes place during the last state of the instruction cycle in which the request occurs, thus ensuring that any instruction in progress is completed before the interrupt can be processed.

The INTA (Interrupt Acknowledge) machine cycle which follows the arrival of an enabled interrupt request resembles an ordinary FETCH machine cycle in most respects. The contents of the program counter are latched onto the address lines during T1, but the counter itself is not incremented during the INTA machine

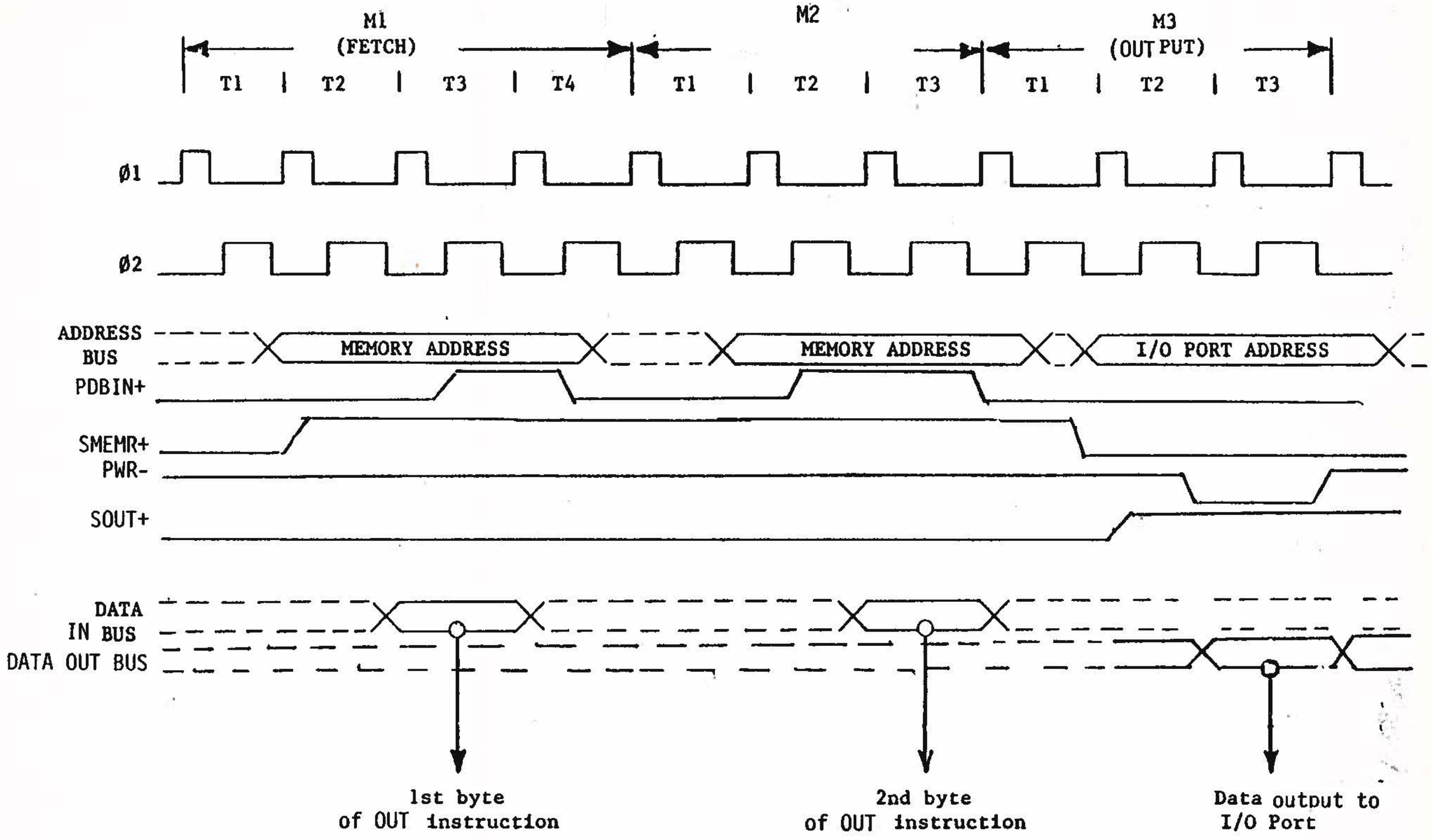


FIGURE 4 . OUTPUT INSTRUCTION CYCLE

cycle, as it otherwise would be. In this way, the pre-interrupt status of the program counter is preserved, so that data in the counter may be saved in the stack. This in turn permits an orderly return to the interrupted program after the interrupt request has been processed.

The INTA output is used to gate an "interrupt instruction" onto the system data bus during state T3. The interrupt instruction is a one-byte RST which causes program control to branch to an interrupt routine. The RST interrupt instruction is supplied by the priority encoder.

2.1.2 HALT SEQUENCES

When a halt instruction (HLT) is executed, the 8080 enters the halt state after state T2 of the next machine cycle. There are only three ways in which the 8080 can exit the halt state:

- A low on the reset input (PRESET) will always reset the 8080 to state T1; reset also clears the program counter.
- An interrupt will cause the 8080 to exit the halt state and enter state T1 on the rising edge of the next 01 clock pulse.

Note: The interrupt enable (INTE) flag must be set when the halt state is entered; otherwise, the 8080 will only be able to exit via a reset signal.

2.1.3 START-UP SEQUENCE

When power is applied initially to the 8080, the processor begins operating immediately. The slow transition of the power supply rise is sensed by an internal Schmitt Trigger which produces a synchronized RESET signal which restores the processor's internal program counter to zero. Program execution thus begins with memory location zero.

2.1.4 8080 CENTRAL PROCESSOR ARCHITECTURE

The next few paragraphs discuss the functional structure of the 8080 Central Processor Unit. Throughout this discussion we have assumed that you have had some experience with assembly language programming; the purpose of Section 2.1.4 is merely to familiarize you with the particular structure of the 8080.

The 8080 CPU contains six general purpose registers which can either be used as single registers (containing 8 bits of data) or as register pairs (containing 16 bits of data). These registers are: B, C, D, E, H, and L. (The register pairs are BC, DE, and HL). Single bytes of data (8-bits) can be transferred between these registers, and between these registers and memory. In addition, one data byte can be moved from these registers into a special register called the accumulator, and from the accumulator into any register. These general purpose registers are used to store data and memory addresses. When used as pairs, the B, D, and H registers contain the most significant byte of the 16-bit contents of the register pair, and the C, E, and L registers contain the least significant byte.

The accumulator (register A) is a special register used by the arithmetic and logical unit of the computer. Whenever an arithmetic operation is performed on data, the data to be operated upon is usually placed in the accumulator, the operation performed, and the result of the operation then left in the accumulator.

```
        EXAMPLE:
        SUB     B
```

The above instruction tells the machine to subtract the contents of the B register from the contents of the accumulator, and leave the results in the accumulator. Besides the contents of other registers, you may move the contents of a memory location into the accumulator. (For more information on the 8080 instruction set, see Section 2.1.5, 8080 INSTRUCTION SET.)

Special one-bit registers are called "flags," which signal certain conditions which may arise as a result of arithmetic operations. The contents of these five flags and the contents of the accumulator are contained within a special register called the PSW (Program Status Word).

```
        EXAMPLE:
        PUSH    PSW
```

The above instruction tells the machine to put (or "push") the contents of the flags and the contents of the accumulator onto the "stack."

Another special register is the PC register (Program Counter). This register contains the address of the next program instruction to be executed. Another register, the SP (Stack Pointer) register, contains the address of the most recent stack entry.

You can also use the general purpose register pairs as addressing registers; that is, to point to the memory location whose contents you want to move into a register. For example, the contents of the HL register pair are always used as the address of the memory byte to be accessed when the symbol M appears in an instruction.

```
        EXAMPLE:

        MOV    B,M
```

The above instruction says: move the contents of the memory byte whose address is in the HL register pair into the B register.

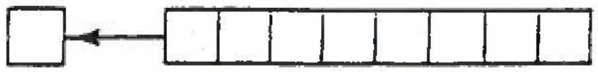
Other instructions use the B and D register pairs to address memory bytes whose contents are to be moved into the accumulator. (Note: these kinds of instructions always move one byte only.)

2.1.5 8080 INSTRUCTION SET

In the 8080 CPU there are five flags.

CARRY FLAG

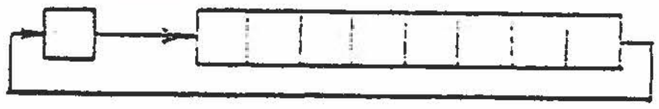
When a number is added to the value in the accumulator, the result may include a carry out of the left-hand bit, the bit of highest significance. This carry "sets" the carry flag to 1.



accumulator

When an addition does not result in a carry out of the most significant accumulator bit, the carry flag is 0.

The carry flag can be set to 0 or 1 by other operations. For instance, the instructions RAR (rotate accumulator right) and RAL (rotate accumulator left) affect the carry flag. In RAR, the least significant bit in the accumulator moves into carry, the bit that was in carry goes into the most significant place in the accumulator, and all other accumulator bits move one place to the right.

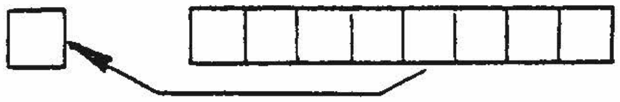


RAL is the opposite of RAR.

The carry flag can be affected by logical operations as well as addition, subtraction, and rotation.

AUXILIARY CARRY FLAG

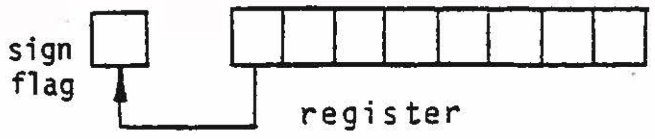
A carry out of the "third bit" (fourth place -- 2^3) sets the auxiliary carry flag:



The auxilary carry flag cannot be tested directly, and exists only to enable the DAA instruction for decimal conversion.

SIGN FLAG

The sign flag is set by certain instructions to duplicate the most significant bit in a register. The most significant bit in a register can be interpreted as the sign of the data quantity when the quantity is considered to be two's complement.



ZERO FLAG

The zero flag is set to 1 at the end of certain operations if the byte resulting from the operations is all zeroes; the zero bit is reset to 0 if the result is not zero.

A result that consists of eight zeroes plus a carry out of the seventh bit sets the zero flag to 1, and also sets the carry flag to 1.

PARITY FLAG

"Parity" refers to whether the number of 1's in a byte is even or odd. Byte parity is checked after certain operations. If the number is even, parity is "even" and the parity flag is set to 1; if there is an odd number of 1's, parity is "odd" and the parity flag is reset to 0.

INSTRUCTIONS

Following is a complete list with discussion of all the operations built into the Poly central processor. The discussion divides the operations into groups of related instructions. Each operation is identified by a "mnemonic" which corresponds to an instruction in machine language ("opcode"). For a chart showing all assembly mnemonics and the associated opcodes, see appendix.

CARRY FLAG INSTRUCTIONS. Two instructions affect the carry flag alone:

CMC (complement carry). Complement the carry flag -- Set it to 0 if it is 1 or to 1 if it is 0.

STC (set carry). Set the carry bit to 1.

SINGLE REGISTER INSTRUCTIONS. These instructions affect the contents of one memory address or any one of the CPU registers -- one byte. If memory, the instruction affects the byte addressed by pair H.

INR (increment register or memory). Increment the affected register or memory byte by 1 -- add 1 to it.

DCR (decrement register or memory). Decrement register or memory byte by 1. This instruction is th opposite of INR -- it is identical to it except that it reduces the affected byte by 1. All flags may be affected.

CMA (complement accumulator). Complement the byte in the accumulator -- change every 1 to 0 and 0 to 1. No flags are affected.

DAA (decimal adjust accumulator). Adjust the byte in the accumulator to form two groups of four bits, each representing one decimal digit. This instruction is rather complicated, treating as it does the awkard relationship between binary and decimal. It is used -- infrequently -- when a decimal output is desired. DAA adjusts the first four bits and second four bits of the accumulator byte separately. First, the less significant four bits of the accumulator byte are compared to 1001 to see if they are greater than nine. If they are (or if the auxiliary carry flag is set to 1), then the accumulator is incremented by six -- which reduces the value of the four bits to nine or less. Next, if the four more significant bits of the accumulator byte represent a number greater than nine (or if the carry flag is set to 1), then these four bits are incremented by six, so that they will represent a value of nine or less. Note that either of these two adjustments may have produced a carry. A carry out of the four less significant bits sets the auxiliary carry flag to 1; otherwise, it is reset. A carry out of the accumulator byte sets the carry flag to 1; otherwise, it retains its previous value. All other flags may be affected.

NO-OPERATION INSTRUCTION:

One instruction results in no operation.

NOP (no operation). Move on to the next instruction in sequence. No flags are affected.

DATA TRANSFER INSTRUCTIONS:

These instructions transfer data between registers or between memory and registers.

MOV (move). Move one byte of data from an indicated register or memory to another individual register or memory. The data also remains in its original location.

Format example: MOV B,A. "Move the byte in A (accumulator) into register B." Note that the format states the affected register first. Data cannot be moved from one memory address to another in a single operation. Data moved out of memory is always taken from the location addressed by H & L. No flags are affected.

STAX (store accumulator). Store the contents of the accumulator into the memory location addressed by register pair B or pair D. No flags are affected.

LDAX (load accumulator). Store the contents of the memory location addressed by the indicated register pair (pair B or pair D) into the accumulator. No flags are affected.

REGISTER OR MEMORY TO ACCUMULATOR INSTRUCTIONS.

These instructions operate on the accumulator using a byte taken from a register or from memory. Memory is taken from the memory location addressed by the data pointer (H & L). Results are left in the accumulator.

ADD (add register or memory to accumulator). Add the byte in one register or in memory to the value in the accumulator. ADD A doubles the accumulator. All flags may be affected.

ADC (add register or memory and carry flag bit to accumulator). Add the byte from a specified location, plus the value of the carry flag, to the value in the accumulator. All flags may be affected.

SUB (subtract register or memory from accumulator). Subtract the byte in a specified register or memory location from the value in the accumulator. SUB A subtracts the accumulator from itself, leaving it (and the carry flag) at zero. All flags may be affected.

SBB (subtract register or memory and carry flag bit -- "borrow" -- from accumulator). Subtract the byte taken from a specified location, plus the value of the carry flag, from the accumulator. All flags may be affected.

ANA (AND register or memory with accumulator). AND the specified byte with the accumulator. ANA is often used to zero part of the accumulator. Carry, zero, sign, and parity flags may be affected.

XRA (XOR register or memory with accumulator). XOR the specified byte with the value in the accumulator. XRA A zeroes the accumulator. Then a MOV from A to a register zeroes that register. All flags may be affected.

ORA (OR register or memory with accumulator). OR the specified byte with the value in the accumulator. This instruction is often used to set part of the accumulator to 1's. Flags affected: carry flag is zeroed; zero, sign, and parity flags may be affected.

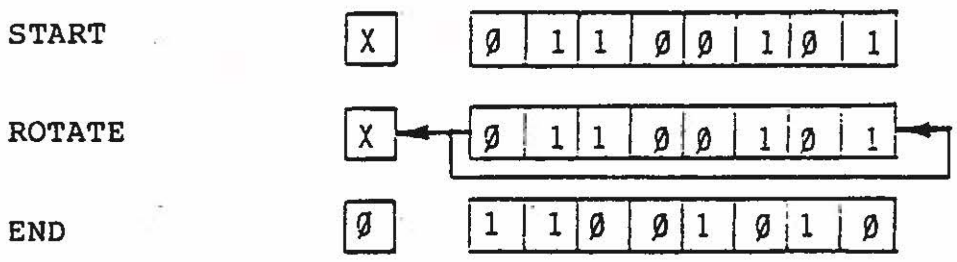
CMP (compare register or memory with accumulator). Compare the specified byte to the contents of the accumulator. In effect, this determines if the specified byte is smaller than, equal to, or larger than the accumulator byte. Flags: the zero flag is 1 if the quantities are equal, and 0 if they are unequal. The carry flag is 1 if the register or memory byte is larger than the accumulator byte, and 0 otherwise (but when the two compared values differ in sign, the sense of the carry flag is reversed). All other flags may also be affected.

ROTATE ACCUMULATOR INSTRUCTIONS.

These instructions rotate the contents of the accumulator -- move a bit from one end, and shift the other bits one place. Rotation can be to the left or to the right, and involves the carry flag bit (but no other).

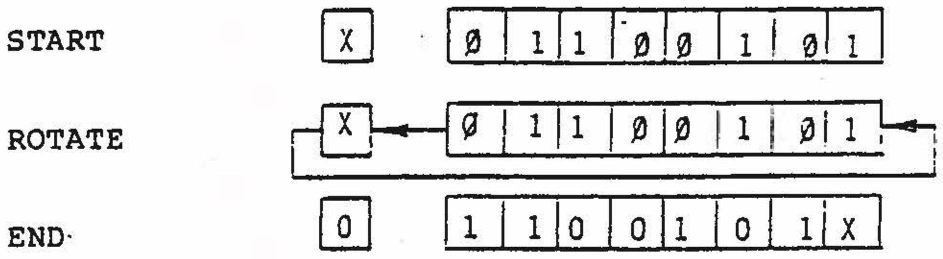
RLC (rotate accumulator left and into carry). Move the most significant bit in the accumulator (left-hand bit) into the carry flag and into the least significant place in the accumulator.

All other bits shift one place to the left.



RRC (rotate accumulator right and into carry). Move the least signficant bit from the accumulator into carry and into the most significant place; the opposite of the instruction above.

RAL (rotate accumulator left, through carry). Move the most significant accumulator bit into carry, and the carry flag bit into the least significant place; shift all other accumulator bits left.

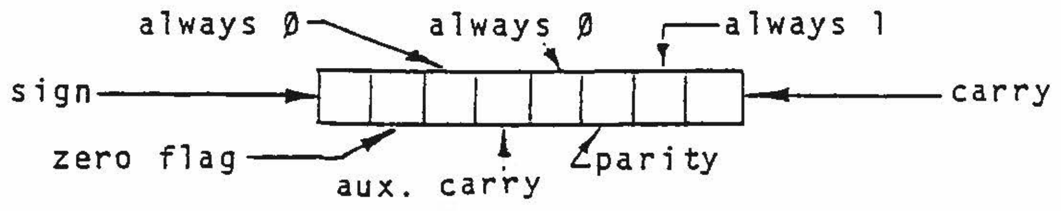


RAR (rotate accumulator right, through carry). Move the least significant bit to carry, and move carry into the most significant place; the opposite of the instruction above.

REGISTER PAIR INSTRUCTIONS.

These instructions operate on the register pairs.

PUSH (push data onto stack). Store the value in the specified register pair into the two bytes of memory addressed by the stack pointer. Such data is said to be "pushed" onto "the stack." The more significant byte goes into address SP-1, the less significant into address SP-2. Indicating PSW (processor status word) stores the current accumulator value at SP-1 and a byte incorporating all the flags in SP-2:



The stack pointer is left pointing to the address where the second byte has been stored. Flags are not affected.

POP (pop data off stack). Store data from the stack into the indicated register pair. The byte of data at SP is stored into the less significant register; the byte at SP+1 goes into the more significant register. If register pair PSW is indicated, the byte at SP provides the bits of the flags, and the byte at SP+1 goes into the accumulator. This instruction is the opposite of the one above.

DAD (double add). Add the two-byte value in the indicated register pair (B, D, or H) to the two-byte value in pair H, and leave the result in pair H. Flag affected: carry.

INX (increment extended register pair). Increment the value in a register pair by 1 -- add 1 to it. No flags are affected.

DCX (decrement extended register pair). Subtract 1 from a register pair. The opposite of the above. No flags are affected.

XCHG (exchange registers). Move the value in pair H to pair D and vice versa. No flags are affected.

XTHL (exchange H & L with stack). Exchange the value in L with the value in the memory location addressed by the stack pointer and exchange the value in H with the value in that memory address plus one (SP plus 1). No flags are affected.

SPHL (load SP from H & L). Load the value in register pair H into the stack pointer register. That value is now the stack address pointed to by the stack pointer. No flags are affected.

IMMEDIATE INSTRUCTIONS.

These instructions operate on one or two bytes of data, included in the instruction itself. The data immediately follows the opcode (hence "immediate").

LXI (load extended immediate). Load the indicated register pair with the two bytes immediately following. The first byte goes into the lower-order register, the second into the higher-order register. No flags are affected.

MVI (move immediate). Move the following byte into the specified register or into the memory location addressed by the data pointer. This instruction resembles LXI except that it enters only one byte of data (and therefore can be used to load a memory location. No flags are affected.

ADI (add immediate to accumulator). Add the following byte to the value in the accumulator, and leave the result in the accumulator. All flags may be affected.

ACI (add immediate, plus the carry bit, to accumulator). Add the following byte, plus the value of the carry flag bit, to the value in the accumulator, and leave the result in the accumulator. All flags may be affected.

SUI (subtract immediate from accumulator). Subtract the following byte from the value in the accumulator, and leave the result in the accumulator. All flags may be affected. This instruction is the subtraction equivalent of ADI above.

SBI (subtract immediate, and "borrow," from accumulator). Subtract the byte immediately following, and the value of the carry flag bit, from the value in the accumulator, and leave the result in the accumulator. This is the subtraction equivalent of ACI above. All flags may be affected.

ANI (AND immediate with accumulator). AND the byte immediately following with the value in the accumulator, and leave the result in the accumulator. Carry, zero, sign, and parity flags may be affected.

XRI (XOR immediate with accumulator). XOR the byte immediately following with the value in the accumulator, and leave the result in the accumulator. The carry flag is set to 0. Zero, sign, and parity flags may also be affected.

ORI (OR immediate with accumulator). OR the byte immediately following with the value in the accumulator, and leave the result in the accumulator. The carry flag is set to 0. Zero, sign, and parity flags may also be affected.

CPI (compare immediate data with accumulators). Compare the following byte to the value in the accumulator. The zero flag is set to 1 if the two values are equal and 0 if they are unequal. The carry flag is set to 1 if the immediate data value is larger than the accumulator value, and set to 0 otherwise. (But if the two values differ in sign, the sense of the carry flag is reversed.) All other flags may be affected.

DIRECT ADDRESSING INSTRUCTION.

These instructions involve the contents of memory addresses; the addresses are included as part of the instruction. The instruction states the address "backwards" -- first the less significant address byte, then the more significant. These instructions do not affect flags.

STA (store accumulator direct). Store the value in the accumulator into the memory location addressed in the instruction.

LDA (load accumulator direct). Load the contents of the memory location addressed in the instruction into the accumulator. No flags are affected. This instruction is the opposite of STA above.

SHLD (store H and L direct). Store the contents of register pair H into the memory location addressed in the instruction. No flags are affected.

LHLD (load accumulator direct). Load the contents of the memory location addressed by the instruction into the L register, and the contents of the next higher address into the H register. This is the opposite of SHLD above.

JUMP INSTRUCTIONS

These instructions cause the computer to "jump" to another part of a program rather than continue to perform instructions in sequence. None of these instructions affects flags.

PCHL (load program counter with H & L). Load the contents of register H into the more significant byte of te program counter, and the contents of register L into the less significant byte. The next instruction executed will be the one now addressed by the program counter. Note that this instruction does not itself contain an address. All other jump instructions do.

JMP (jump). Execute the instruction located at the address given in the instruction, and continue sequentially. This is called an "unconditional jump." All the following jump instructions are "conditional."

JC (jump if carry). Jump to the instruction addressed by this instruction if the carry flag is set to 1. If the carry flag is 0, move on to the next instruction in sequence.

JNC (jump if no carry). Jump to the instruction addressed by this instruction if the carry flag is set to 0. If the carry flag is 1, move on to the next instruction in sequence. This instruction is the opposite of the above.

JZ (jump if zero). Jump to the instruction addressed by this instruction if the zero flag is set to 1. If the zero flag is set to 0, move on to the next instruction in sequence. Compare to JC. (Note that the "if zero" condition is met if the register in question is all zeroes, so that the zero flag is set to 1.)

JNZ (jump if not zero). Jump to the instruction addressed by this instruction if the zero flag is set to 0. If the zero flag is set to 1, move on to the next instruction in sequence. This instruction is the opposite of JZ above. Compare to JNC.

JM (jump if minus). Jump to the instruction addressed by this instruction if the sign flag is set to 1 ("minus"). If the sign flag is set to 0, move on to the next instruction in sequence. Compare to JC and JZ above.

JP (jump if plus). Jump to the instruction addressed by this instruction if the sign flag is set to 0 ("plus"). If the sign flag is set to 1, move on to the next instruction in sequence. This instruction is the opposite of JM above. Compare to JNC and JNZ.

JPE (jump if parity even). Jump to the instruction addressed by this instruction if the parity flag is set to 1 ("even parity"). If it is set to 0, move on to the next instruction in sequence. Compare to JC, JZ, and JM above.

JPO (jump if parity odd). Jump to the instruction addressed by this instruction if the parity flag is set to 0 ("parity odd").

If the parity flag is 1, move on to the next instruction in sequence. This instruction is the opposite of JPE above. Compare to JNC, JNZ, and JP above.

CALL SUBROUTINE INSTRUCTIONS

Like jump instructions, call instructions cause the computer to depart from sequential execution of instructions. Also like jump instructions, most are "conditional" -- most operate only if some condition is met. And as with jump instructions, execution of instructions continues in sequence starting with the instruction at the address called (stated in the call instruction). The two types of instructions also resemble one another in that the address included is stated "backwards" -- first the less significant address byte, then the more significant. Also, these instructions do not affect flags.

The two kinds of instructions differ in that a call instruction "pushes" an address onto "the stack" -- namely, the address of the instruction to which the computer will "return" when it has finished the subroutine.

CALL. Go to the subroutine addressed by this instruction, and begin sequential execution there. This is an "unconditional call," and corresponds to an unconditional jump. All other call instructions are conditional, and correspond to the conditional jump instructons, each triggered by the state of one of the flags.

CC (call if carry). Go to the subroutine addressed by this instruction if the carry flag is set to 1. If the carry flag is 0, move on to the next instruction in sequence.

CNC (call if no carry). Go to the subroutine addressed in this instruction if the carry flag is set to 0. If the carry flag is 1, move on to the next instruction in sequence. This instruction is the opposite of CC above.

CZ (call if zero). Go to the subroutine addressed by this instrucion of the zero flag is set to 1. If the zero flag is 0, move on to the next instruction in sequence. Compare to CC.

CNZ (call if not zero). Go to the subroutine addressed by this instruction if the zero flag is set to 0. If the zero flag is 1, move on to the next instruction in sequence. This instruction is the opposite of CZ above. Compare to CNC.

CM (call if minus). Go to the subroutine addressed by this instruction if the sign flag is set to 1 ("minus"). If the sign flag is 0, move on to the next instruction in sequence. Compare to CC and CZ above.

CP (call if plus). Go to the subroutine addressed by this instruction if the sign flag is set to 0 ("plus"). If the sign flag is 1, move on to the next instruction. This instruction is the opposite of CM above. Compare to CNC and CNZ above.

CPE (call if parity even). Go to the subroutine addressed by this instruction if the parity flag is set to 1 ("even parity"). If the parity flag is 0, move on to the next instruction in sequence. Compare to CC, CZ, and CM above.

CPO (call if parity odd). Go to the subroutine addressed in this instruction if the parity flag is set to 0. If the parity flag is 1, move on to the next instruction. This instruction is the reverse of CPE above. Compare to CNC, CNZ, and CP above.

RETURN FROM SUBROUTINE INSTRUCTIONS.

These instructions get the computer back from subroutines to the instruction following the call instruction that caused it to depart. Specifically, they "pop" an address previously "pushed" onto "the stack" off of the stack and into the program counter, causing the computer to next execute the instruction located at that address. Execution then continues sequentially from there. Each return instruction is associated with a previous call insruction, i.e. the program counter always returns eventually to the point in a program that it previously departed from (to an instruction following a call instruction). Therefore the number of returns executed is always equal to the number of calls executed is always equal to the number of calls executed (unless the machine halts).

Since these instructions always "pop" addresses in the order opposite that in which they were "pushed," they can be said always to operate on the "next available address" in the stack, so that the address need not be stated in the instruction.

Like "jump" and call instructions, all but one of the return instructions are conditional upon the state of one of the flags. Flags are not affected by return instructions.

RET (return). Return to the most recently pushed address. This is an "unconditional return."

RC (return if carry). Return to the next address on the stack if the carry flag is 1. If the carry flag is 0, move on to the next instruction in sequence.

RNC (return if no carry). Return to the next address on the stack if the carry flag is 0. If the carry flag is 1, move on to the next instruction in sequence. This instruction is the opposite of RC above.

RZ (return if zero). Return to the next address on the stack if the zero flag is 1. If the zero flag is 0, move on to the next instruction in sequence. Compare to RC above.

RNZ (return if not zero). Return to the next address on the stack if the zero flag is 0. If the zero flag is 1, move on to the next instruction in sequence. This instruction is the opposite of RZ above. Compare to RNC above.

RM (return if minus). Return to the next address on the stack if the sign flag is 1 ("minus"). If the sign flag is 0, move on to the next instruction in sequence. Compare to RC, RZ above.

RP (return if plus). Return to the next address on the stack if the sign flag is 0 ("plus"). If the sign flag is 0, move on to the next instruction in sequence. This instruction is the opposite of the instruction above. Compare to RNC, RNZ above.

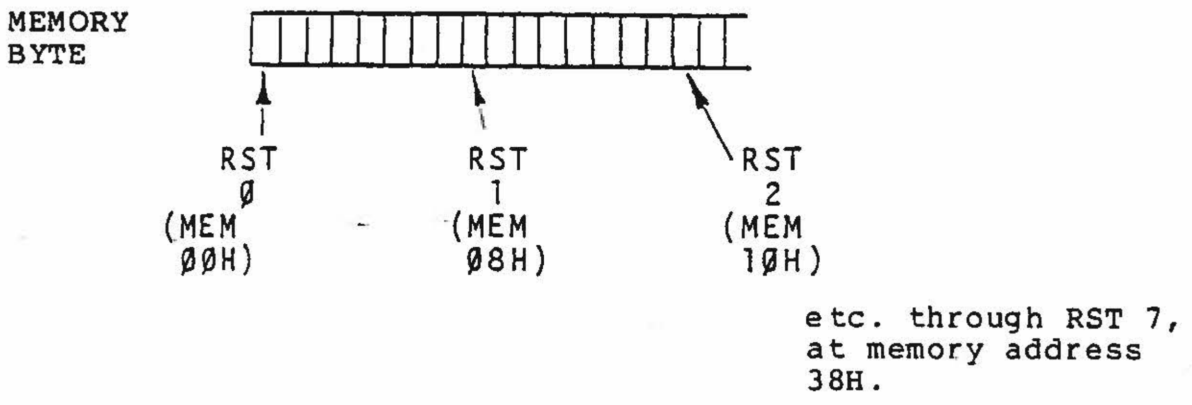
RPE (return if parity even). Return to the next address on the stack if the parity flag is 1 ("even parity"). If the parity flag is 0, move on to the next instruction in sequence. Compare to RC, RZ, RM above.

RPO (return if parity odd). Return to the next address on the stack if the parity flag is 0 ("odd parity"). If the parity flag is 1, move on to the next instruction in sequence. This instruction is the opposite of RPE above. Compare to RNC, RNZ, RP above.

RESTART INSTRUCTION.

One special instruction, RST, resembles the call instructions in that it pushes a return address onto the stack and sends the computer off to another location. The address of the instruction following the RST instruction sequentially is pushed onto the stack, so that the computer will eventually return to its point of departure. Note that the RST instruction pushes the address of the instruction following RST -- otherwise the computer would return to the RST instruction itself and be trapped in an endless loop.

RST sends the computer (i.e. the program counter) off to one of eight pre-determined memory locations, each the first of a sequence of eight bytes, making up the first sixty-four bytes of memory.



Actually, the eight bytes associated with each RST can be reached by means of other kinds of instructions -- jump and call instructions -- and need not comprise individual routines. In the POLY 88, all sixty-four of these bytes are used in the monitor (discussed later).

The CPU executes an RST at one of two times. An RST instruction may be written into a program, in which case the instruction is in effect a "call" instruction in shorter form -- one byte instead of three. More usually, the CPU executes an RST when the running of a program is interrupted "from the outside". For instance, saving onto tape is a very slow process for the POLY 88, which can output data much faster than the tape recorder can properly record it. So the computer outputs data to the tape on an interrupt basis -- it occupies itself with other tasks until the output port electronics indicate that it is time to output another data item to the tape. This forces a restart, which puts a book marker into the program so the computer will be able to get back to its point of departure, and sends the program counter off to a pre-determined location to begin execution of a brief routine that causes the computer to output a data item to the tape.

INTERRUPT FLIP-FLOP INSTRUCTIONS.

Sometimes it is important not to permit interruptions of a program. For that reason, interrupts can be disabled--input or output electronics can be prevented from forcing a restart. Whether or not interrupts are disabled depends upon the state of a single flip-flop, called the interrupt flip-flop is reset to 0, from which time interrupts are disabled till the flip-flop is once again set to 1. No flags are affected.

EI (enable interrupt). Set the interrupt flip-flop to 1.

DI (disable interrupt). Reset the interrupt flip-flop to 0.

INPUT/OUTPUT INSTRUCTIONS.

These instructions cause the computer to input data from or output data to a device external to the computer -- like a keyboard.

To be precise, the instruction causes the CPU to open an input or output port, which is assumed to provide a connection with some device. No flags are affected.

IN (input). Load one byte from the designated input port into the accumulator.

OUT (output). Send the byte in the accumulator out to the designated output port.

HALT INSTRUCTION.

This instruction brings computer operations to a stop. It first increments the program counter -- adds 1 to it -- so that the computer will resume with the next instruction. No flags are affected.

HLT (halt). Increment the program counter, then stop.

2.2 CPU USART OPTION

The USART is included on every assembled CPU card. It is an option which may be ordered separately for use with CPU card bits.

The USART option for the 8080 CPU card provides a very flexible serial communications interface for the POLY 88 and system 88. The USART option and a software controllable baud rate generator. The USART may interface to two serial I/O devices. The device and its baud rate may be changed under the control of one output port. These interface through two "minicards" which mount to the rear of the POLY 88 or system 88 chassis. Interface cards are available for RS-232-C, current loop, Kansas City (Byte) Standard audio cassette, and the 2400 baud Polyphase cassette.

This note describes the USART option from a functional standpoint and then describes the various operating modes of the USART and how the USART may be programmed.

COMMUNICATIONS FORMATS

Serial communications, either on a data link or with a local peripheral, occurs in one of two basic formats: asynchronous or synchronous. These formats are similar in that they both require framing information to be added to the data to enable proper detection of the character at the receiving end. The major difference between the two formats is that the asynchronous format requires framing information to be added to each character, while the synchronous format adds framing information to blocks of data, or messages. Since the synchronous format is more efficient than the asynchronous format but requires more complex decoding it is typically found on high-speed data links, while the asynchronous format is used on lower-speed lines.

The asynchronous format starts with the basic data bits to be transmitted and adds a "START" bit to the front of them and one or more "STOP" bits behind them as they are transmitted. The START bit is a logical zero, or SPACE, and is defined as the positive voltage level by RS-232-C. The STOP bit is a logical one, or MARK, and is defined as the negative voltage level by RS-232-C.

In current loop applications, current flow normally indicates a MARK and lack of current a SPACE. The START bit tells the receiver to start assembling a character and allows the receiver to synchronize itself with the transmitter. Since this synchronization only has to last for the duration of the character (the next character will contain a new START bit), this method works quite well assuming a properly designed receiver. One or more STOP bits are added to the end of the character to ensure that the START bit of the next character will cause a transition on the communication line and to give the receiver time to "catch up" with the transmitter if its basic clock happens to be running slightly slower than the transmitter clock. If, on the other hand, the receiver clock happens to be running slightly faster than the transmitter clock, the receiver wil perceive gaps between characters but will still correctly decode the data. Because of this tolerance to minor frequency deviations, it is not necessary that the transmitter and receiver clocks be locked to the identical frequency for successful asynchronous communication.

The synchronous format, instead of adding bits to each character, groups characters into records and adds framing characters to the record. The framing characters are generally known as SYN characters and are used by the receiver to determine where the character boundaries are in a string of bits. Since synchronization must be held over a fairly long stream of data, bit synchronization is normally either extracted from the communication channel by the modem or supplied from an external source.

An example of the synchronous and asynchronous formats is shown in Figure 5. The synchronous format shown is fairly typical in that it requires two SYN characters at the start of the message.

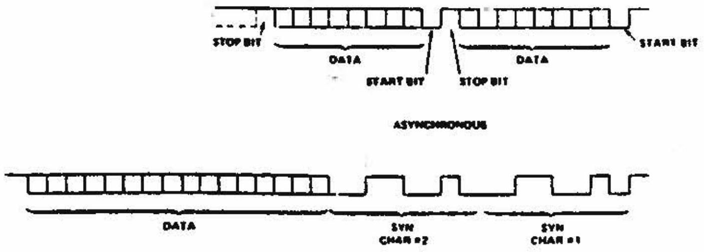


Figure 5.

The asynchronous format, also typical, requires a START bit preceding each character and a single STOP bit following it. In both cases, two 8-bit characters are to be transmitted. In the asynchronous mode, 10 x n bits are used to transmit n characters, and in the synchronous mode, 8n + 16 bits are used. For the example shown, the asynchronous mode is actually more efficient, using 20 bits versus 32. Transmitting a thousand characters in the asynchronous mode, however, takes 10,000 bits, versus 8,016 for the synchronous mode. For long messages, the synchronous format becomes much more efficient than the asynchronous format; the crossover point for the examples shown in Figure 5 is eight characters, for which both formats require 80 bits.

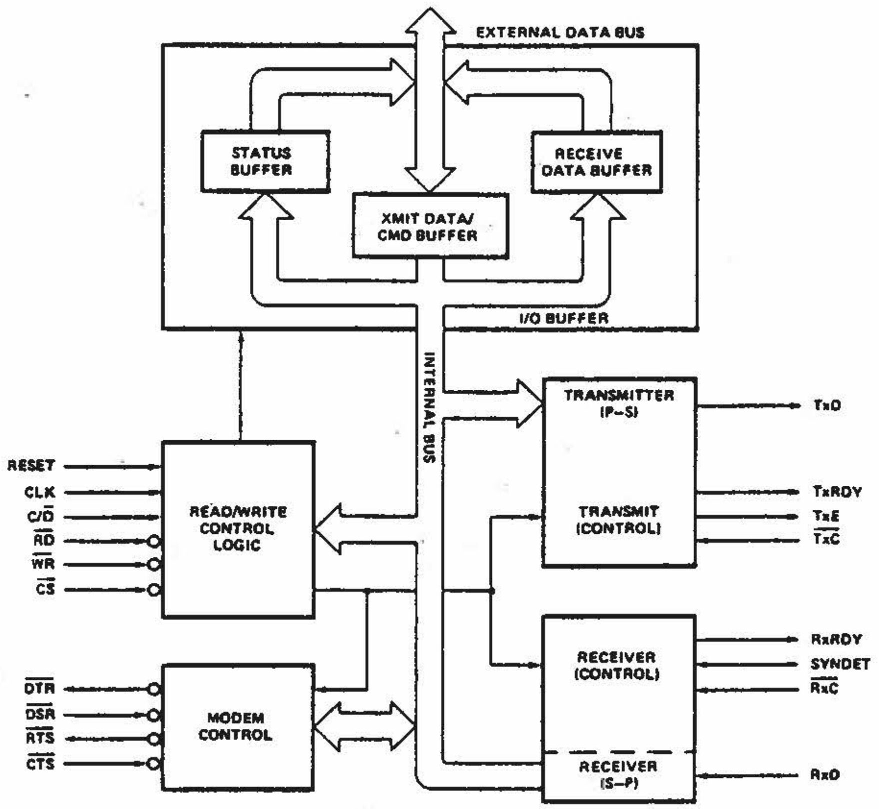
In addition to the differences in format between synchronous and asynchronous communication, there are differences with regards to the type of modems that can be used. Asynchronous modems typically employ Frequency Shift Keying (FSK) techniques which simply generate one audio tone for a MARK and another for a SPACE. The receiving modem detects these tones on the telephone line, converts them to logic signals, and presents them to the receiving terminal. Since the modem itself is not concerned with the transmission speed, it can handle baud rates from zero to its maximum speed. Synchronous modems, in contrast to asynchronous modems, supply timing information to the terminal and require data to be presented to them synchronized with this timing information. Synchronous modems, because of this extra clocking, are only capable of operating at certain preset baud rates. The receiving mode, which has an oscillator running at the same frequency as the transmitting modem, phase-locks its clock to that of the transmitter and interprets changes of phase as data. The PolyMorphic PolyPhase cassette interface operates in a synchronous mode.

In some cases it is desirable to operate in a hybrid mode which involves transmitting data with the asynchronous format using a synchronous modem. This occurs when an increase in operating speed is required without a change in the basic protocal of the system. This hybrid technique is known as isosynchronous and involves the generation of the start and stop bits associated with the synchronous format, while still using the modem clock for bit synchronization. The Byte standard cassette interface operates in an isosynchronous mode.

The 8251 USART (Universal Synchronous/Asynchronous Receiver-Transmitter) has been designed to meet a broad spectrum of requirements in the synchronous, asynchronous, and isosynchronous modes. In the synchronous modes it operates with 5, 6, 7, or 8-bit characters. Even or odd parity can be optionally appended and checked. Synchronization can be achieved internally via SYN character detection. SYN detection can be based on one or two characters which may or may not be the same. The single or double SYN characters are inserted into the data stream automatically if the software fails to supply data in time. The automatic generation of SYN characters is required to prevent the loss of synchronization. In the asynchronous mode the USART operates with the same data and parity structures as it does in the synchronous mode. In addition to appending a START bit to this data, it apprends 1, 1 1/2, or 2 STOP bits. Proper framing is checked by the receiver and a status flag set if an error occurs. In the asynchronous mode the USART can be programmed to accept clock rates of 1, 16, or 64 times the required baud rate. Note that X1 operation is only valid if the clocks of the receiver and transmitter are synchronized.

The USART can transmit the three formats in half or full duplex mode and is double-buffered internally (i.e., the software has a complete character time to respond to a service request). Although the USART supports basic data set control signals (DTR, RST, etc.), it does not fully support the signaling described in EIA RS-232-C. Examples of unsupported signals are Ring Indicator (CE), and the second channel signals. The serial option does not interface to the voltage levels required by EIA RS-232-C; this function is provided by the Printer Interface Card. (This card also provides an optically isolated current loop interface.)

A block diagram of the USART is shown in Figure 6. As can be seen in the figure, the USART consists of five major sections which communicate with each other on an internal data bus. The five sections are the receiver, transmitter, modem control, read/write control, and I/O buffer. In order to facilitate discussion, the I/O buffer has been shown broken down into its three major subsections: the status buffer, the transmit data/command buffer, and the receive data buffer.



RECEIVER

The receiver accepts serial data on the RxD pin and converts it to parallel data according to the appropriate format. When the

USART is in the asynchronous mode, and it is ready to accept a character (i.e., it is not in the process of receiving a character), it looks for a low level on the RxD line. When it sees the low level, it assumes that it is a START bit and enables an internal counter. At a count equivalent to one-half of a bit time, the RxD line is sampled again. If the line is still low, a valid START bit has probably been received and the USART proceeds to assemble the character. If the RxD line is high when it is sampled, then either a noise pulse has occurred on the line or the receiver has become enabled in the middle of the transmission of a characcter. In either case, the receiver aborts its operation and prepares itself to accept a new character.

After the successful reception of a START bit, the USART clocks in the data, parity, and STOP bits, and then transfers the data on the internal data bus to the receive data register. When operating with less than 8 bits, the characters are right-justified. The RxRDY signal is asserted to indicate that a character is available.

In the synchronous mode, the receiver simply clocks in the specified number of data bits and transfers them to the receiver buffer register, setting RxRDY. Since the receiver blindly groups data bits into characters, there must be a means of synchronizing the receiver to the transmitter so that the proper character boundaries are maintained in the serial data stream. This synchronization is achieved in the HUNT mode.

In the HUNT mode the USART shifts in data on the RxD line one bit at a time. After each bit is received, the receiver register is compared to a register holding the SYN character (program loaded). If the two registers are not equal, the USART shifts in another bit and repeats the comparison. When the registers compare as equal, the USART ends the HUNT mode and raises the SYNDET line to indicate that it has achieved synchronization. If the USART has been programmed to operate with two SYN characters, the process is as described above, except that two contiguous characters from the line must compare to the two stored SYN characters before synchronization is declared. Parity is not checked. The USART enters the HUNT mode when it is initialized into the synchronous mode or when it is commanded to do so by the command instruction. Before the receiver is operated, it must be enabled by the RxE bit (D2) of the command instructions. If this bit is not set, the receiver will not assert the RxRDY bit.

TRANSMITTER

The transmitter accepts parallel data from the processor, adds the appropriate framing information, serializes it, and transmits it on the TxD pin. In the asynchronous mode the transmitter always adds a START bit; depending on how the unit is programmed, it also adds an optional even or odd parity bit, and either 1, 1 1/2, or 2 STOP bits. In the asynchronous mode no extra bits (other than parity, if enabled) are generated by the transmitter unless the computer fails to send a character to the USART. If the USART is ready to transmit a character and a new character has not been supplied by the computer, the USART will transmit a

SYN character. This is necessary since synchronous communications, unlike asynchronous communications, does not allow gaps between characters. If the USART is operating in the dual SYN mode, both SYN characters will be transmitted before the message can be resumed. The USART will not generate SYN characters until the software has supplied at least one character; i.e., the USART will "fill holes" in the transmission but will not initiate transmission itself. The SYN characters which are to be transmitted by the USART are specified by the software during the initialization procedure. In either the synchronous or asynchronous modes, transmission is inhibited until TxEnable and the -CTS input are asserted.

An additional feature of the transmitter is the ability to transmit a BREAK. A BREAK is a period of continuous SPACE on the communication line and is used in full duplex communication to interrupt the transmitting terminal. The USART will transmit a BREAK condition as long as bit 3 (SBRK) of the command register is set.

MODEM CONTROL

The modem control section provides for the generation of -RTS and the reception of -CTS. In addition, a general-purpose output and a general-purpose input are provided. The output is labeled -DTR and the input is labeled -DSR. -DTR can be asserted by setting bit 2 of the command instruction; -DSR can be sensed as bit 7 of the status register. Although the USART itself attaches no special significance to these signals, -DTR (Data Terminal Ready) is normally assigned to the modem, indicating that the terminal is ready to communicate and -DSR (Data Set Ready) is a signal from the modem indicating that it is ready for communications.

I/O CONTROL

The Read/Write Control Logic decodes control signals on the 8080 control bus into signals which gate data on and off the USART's internal bus and controls the external I/O bus (DB0 DB7). The receiver and transmitter buffers are located at port #0, while the status and command buffers are port #1. The I/O buffer contains the STATUS buffer, the RECEIVE DATA buffer, and the TRANSMIT DATA/COMMAND buffer, as shown in Figure 7. Note that although there are two registers which store data for transfer to the CPU (STATUS and RECEIVE DATA), only one register stores data being transferred to the USART. The sharing of the input register for both transmit data and command makes it important to ensure that the USART does not have data stored in this register before sending a command to the device. The TxRDY signal can be monitored to accomplish this. Neither data nor commands should be transferred to the USART if TxRDY is low. Failure to perform this check can result in erroneous data being transmitted.

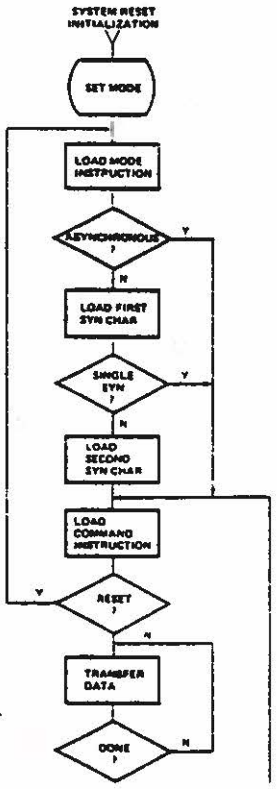
CE	C/$\overline{D}$	$\overline{READ}$	$\overline{WRITE}$	Function
0	0	0	1	CPU Reads Data from USART
0	1	0	1	CPU Reads Status from USART
0	0	1	0	CPU Writes Data to USART
0	1	1	0	CPU Writes Command to USART
1	X	X	X	USART Bus Floating (NO-OP)

Figure 7.

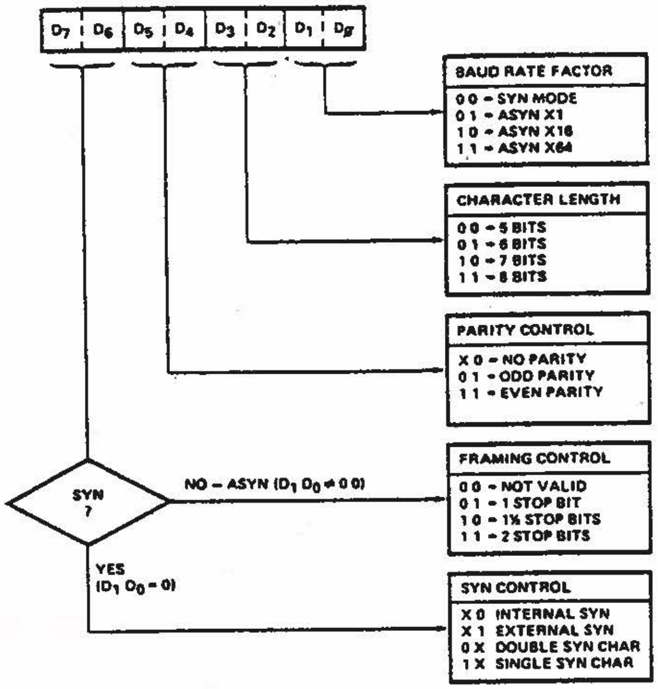
MODE SELECTION

The USART is capable of operating in a number of modes (e.g., synchronous or asynchronous). In order to keep the hardware as flexible as possible (both at the chip and end product levels), these operating modes are selected via a series of control outputs to the USART. These mode control outputs must occur between the time the USART needs this information to structure its internal logic it is essential to complete the initialization before any attempts are made at data transfer (including reading status).

A flowchart of the initialization process appears in Figure 8.



The first operation which must occur following a reset is the loading of the mode control register. The mode control register is loaded by the first control output following a reset. The format of the mode control instruction is shown in Figure 9.



The instruction can be considered as four 2-bit fields. The first 2-bit field (D1 D0) determines whether the USART is to operate in the synchronous (00) or asynchronous mode. In the asynchonous mode this field also controls the clock scaling factor. As an example, if D1 and D0 are both ones, the -RxC and -TxC will be divided by 64 to establish the baud rate. The second field, D3 D2, determines the number of data bits in the character, and the third, D5 D4, controls parity generation. Note that the parity bit (if enabled) is added to the data bits and is not considered as part of them when setting up the character length. As an example, standard ASCII transmission, which is seven bits plus even parity, would be specified as:

X X 1 1 1 0 X X

The last field, D7 D6, has two meanings, depending on whether operation is to be in the synchronous or asynchronous mode. For the asynchronous mode (i.e., D1 D0 = 00), it controls the number of STOP bits to be transmitted with the character. Since the receiver will always operate with only one STOP bit, D7 and D6 only control the transmitter. In the synchronous mode (D1 D0 = 00), this field controls the synchronization process. Note that the choice of single or double SYN characters is independent of the choice of internal or external synchronization. This is because even though the receiver may operate with external synchronization logic, the transmitter must still know whether to send one or two SYN characters should the CPU fail to supply a character in time.

Following the loading of the mode instruction, the appropriate SYN character (or characters) must be loaded if synchronous mode has been specified. The SYN character(s) are loaded by the same control output instruction used to load the mode instruction. The USART determines from the mode instruction whether no, one, or two SYN characters are required and uses the control output to load SYN characters until the required number are loaded.

USART PROGRAMMING HINTS

1. Output of a command to the USART can destroy the integrity of a transmission in progress if timed incorrectly.

   Sending a command into the USART will overwrite any character which is stored in the buffer waiting for transfer to the parallel-to-serial converter in the device. This can be avoided by not sending a command if transmission is taking place. Due to the internal structure of the USART, it is also possible to disturb the transmission if a command is sent while SYN character is being generated by the device. (The USART generates a SYN if the software fails to respond to TxRDY.) If this occurence is possible in a system, commands should be transferred only when a positive-going edge is detected on the TxRDY line.

2. RxE only acts as a mask to RxRDY; it is possible for the USART internal data buffers to already contain one or two characters. These characters should be read and discarded when the RxE bit is first set. Because of these extraneous characters, the proper sequence for gaining synchronization is as follows:

   1. Disable interrupts.

   2. Issue a command to enter hunt mode, clear errors, and enable the receiver (EH,ER,RxE=1).

3. Read USART data port twice(it is not necessary to check status).

4. Enable interrupts.

The first RxRDY that occurs after the above sequence will indicate that the SYN character or characters have been detected and the next character has been assembled and is ready to be read.

3. Loss of CTS or dropping TxEnable will immediately clamp the serial output line.

TxEnable and RTS should remain asserted until the transmission is complete. Note that this implies that not only has the USART completed the transfer of all bits of the last character, but also that they have cleared the modem. A delay of 1 msec following a proper occurence of TxEmpty is usually sufficient (see Item 4). An additional problem can occur in the synchronous mode because the loss of TxEnable clamps the data in at a SPACE instead of the normal MARK. This problem, which does not occur in the asynchronous mode, can be corrected by an external gate combining RTS and the serial output data.

4. Extraneous transitions can occur on TxEmpty while data (including USART generated SYNs) is transferred to the parallel-to-serial converter.

This situation can be avoided by ensuring that TxEmpty occurs during several consecutive status reads before assuming that the transmitter is truly in the empty state.

5. A BREAK (i.e., long space) detected by the receiver results in a string of characters which have framing errors.

If reception is to be continued after a BREAK, care must be taken to ensure that valid data is being received; special care must be taken with the last character perceived during a BREAK, since its value, including any framing error associated with it, is indeterminate.

USART ADDRESSING

The transmit and receive buffers are addressed as port 0 on the CPU card if it is set up for operation at 0. The command and status buffers are located at port 1. If your CPU card is not set up for operation at 0, consult the following table. (See Section 3.5 for addressing option description.)

USART OPTION ADDRESSING

Address Jumper	ROM begins at	Data port	Command & Status Port	Baud Rate Generator Latch
J (factory set)	0000	0	1	4

T	8000	80	81	84
S	E000	E0	E1	E4

2.3 BAUD RATE GENERATOR OPERATION

The baud rate generator may be accessed through port #4. The byte output to this port will be latched into the baud rate latch and determines the baud rate, device number, and on-card memory disable or enable. When power is applied to the CPU card or the front panel reset button is pushed, this latch is set to 0. The command format is as follows:

D7, D6	D5	D4	D3, D2, D1, D0
Unused	ROM disable	Device #	Baud Rate

See Section 3.4, Phantom Memory, for a discussion of on-card memory disable.

BAUD RATE

The baud rate field may assume any value 0 through F (hex). Fifteen of these are valid baud rates; 0 disables clock generation. Note that the actual baud rate is determined by the USART mode (x1, x16, x64 clock). See Figure 12 for the available baud rates. Note the synchronous mode always uses an x1 clock.

At completion of the load of SYN characters (or after the mode instruction in the asynchonous mode), a command character is issued to the USART. The command instruction controls the operation of the USART within the basic framework established by the mode instruction. The format of the command instruction is shown in Figure 10. Note that if, as an example, the USART is waiting for a SYN character load and instead is issued an internal reset command, it will accept the command as an SYN character instead of resetting. This situation, which should only occur if two independent programs control the USART, can be avoided by outputting three all zero characters as commands before issuing the internal reset command. The USART indicated its state in a status register which can be read under program control. The format of the status register read is shown in Figure 11.

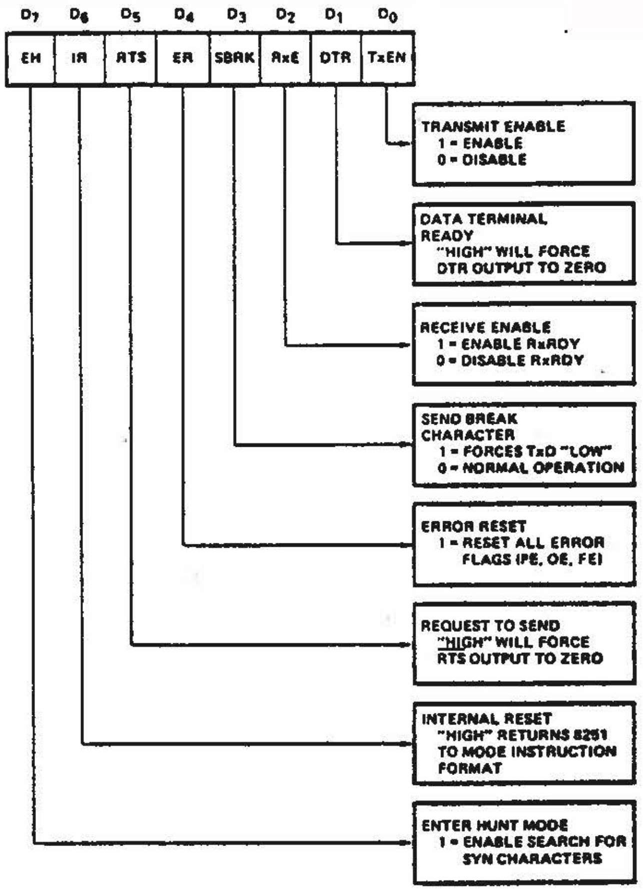

D7 D6 D5 D4 D3 D2 D1 D0
EH IR RTS ER SBRK RxE DTR TxEN
TRANSMIT ENABLE 1 = ENABLE 0 = DISABLE
DATA TERMINAL READY "HIGH" WILL FORCE DTR OUTPUT TO ZERO
RECEIVE ENABLE 1 = ENABLE RxRDY 0 = DISABLE RxRDY
SEND BREAK CHARACTER 1 = FORCES TxD "LOW" 0 = NORMAL OPERATION
ERROR RESET 1 = RESET ALL ERROR FLAGS (PE, OE, FE)
REQUEST TO SEND "HIGH" WILL FORCE RTS OUTPUT TO ZERO
INTERNAL RESET "HIGH" RETURNS 8251 TO MODE INSTRUCTION FORMAT
ENTER HUNT MODE 1 = ENABLE SEARCH FOR SYN CHARACTERS

Figure 10.

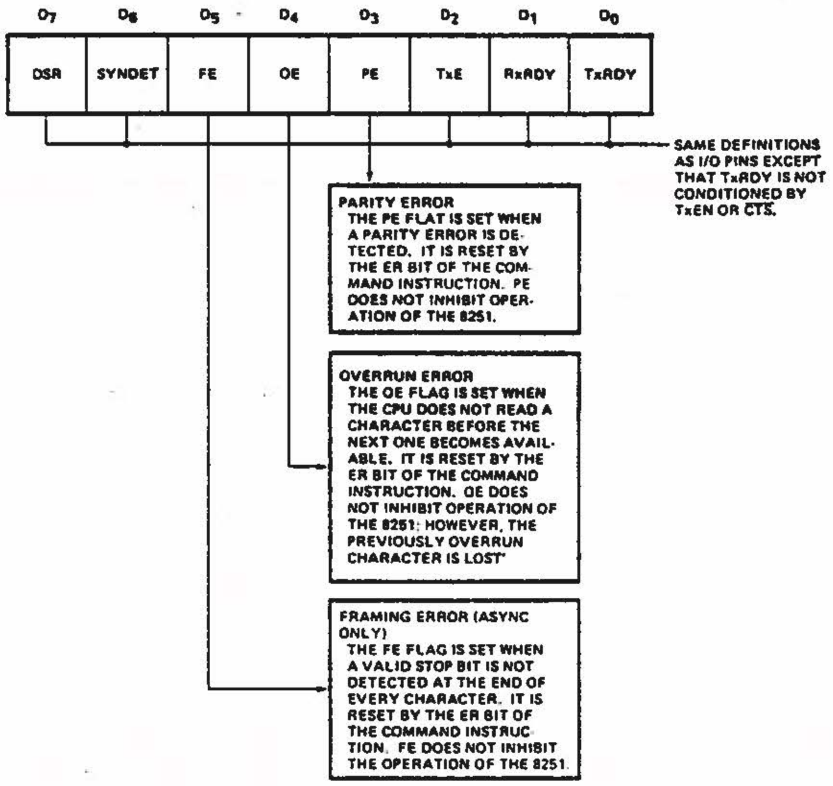

D7 D6 D5 D4 D3 D2 D1 D0
DSR SYNDET FE OE PE TxE RxRDY TxRDY
SAME DEFINITIONS AS I/O PINS EXCEPT THAT TxRDY IS NOT CONDITIONED BY TxEN OR CTS.
PARITY ERROR THE PE FLAT IS SET WHEN A PARITY ERROR IS DETECTED. IT IS RESET BY THE ER BIT OF THE COMMAND INSTRUCTION. PE DOES NOT INHIBIT OPERATION OF THE 8251.
OVERRUN ERROR THE OE FLAG IS SET WHEN THE CPU DOES NOT READ A CHARACTER BEFORE THE NEXT ONE BECOMES AVAILABLE. IT IS RESET BY THE ER BIT OF THE COMMAND INSTRUCTION. OE DOES NOT INHIBIT OPERATION OF THE 8251; HOWEVER, THE PREVIOUSLY OVERRUN CHARACTER IS LOST"
FRAMING ERROR (ASYNC ONLY) THE FE FLAG IS SET WHEN A VALID STOP BIT IS NOT DETECTED AT THE END OF EVERY CHARACTER. IT IS RESET BY THE ER BIT OF THE COMMAND INSTRUCTION. FE DOES NOT INHIBIT THE OPERATION OF THE 8251.

Figure 11.

When operating the receiver, it is important to realize that RxE (bit 2 of the command instruction) only inhibits the assertion of RxRDY; it does not inhibit the actual reception of characters. Because the receiver is constantly running, it is possible for it to contain extraneous data when it is enabled. To avoid problems, this data should be read from the USART and discarded. The read should be done immediately following the setting of Receive Enable in the asynchronous mode,and following the setting of Enter Hunt in the synchronous mode. It is not necessary to wait for RxRDY before executing the dummy read.

USART INTERRUPTS

The SYNDET, TxRDY, and RxRDY flags will cause an interrupt if they are set. These three flags are ORed together and applied to interrupt 3. This connection may be made, if this interrupt is required, by installing a jumper "K" on the CPU card. Two adjacent pads are provided to reconnect the interrupt later, if desired.

When using the PolyMorphic Systems Monitor ROM, the K jumper should be connected.

SER/8 Baud Rates

Baud Rate Field Binary	Hex	USART mode x1	x16	x64
0001	1	800	50	12.5
0010	2	1200	75	18.75
0011	3	1760	110	27.5
0100	4	2152	134.5	33.62
0101	5	2400	150	37.5
0110	6	4800	300	75
0111	7	9600	600	150
1000	8	14400	900	225
1001	9	19200	1200	300
1010	A	28800	1800	450
1011	B	38400	2400	600
1100	C	57600	3600	900
1101	D	-----	4800	1200
1110	E	-----	7200	1800
1111	F	-----	9600	2400

Figure 12.

DEVICE NUMBER

This bit selects the device to be connected to the USART. Two devices (0 and 1) may share the USART on the CPU board. When a device is enabled, it sends data, receive clock, CTS, and DSR to the USART through a tri-state buffer. Transmit data, clock, RTS and DTR are ANDed with the device select signal at the device. Device 0 is normally a cassette interface mini-card and device 1 is normally an RS-232/current loop mini-card.

(Note that the 2 DIP sockets on the CPU do not determine the device number.)

2.4 REAL-TIME CLOCK

The real-time clock (RTC) circuit generates an interrupt for every positive-going edge of the 50 or 60 Hz line frequency. This interrupt is latched and remains on until it is reset by an output command to the real-time clock port. When the programmer services the clock, it is his responsibility to reset the RTC before re-enabling interrupts. This is done by executing an OUT 8, OUT 88H, or OUT 0E8H instruction, depending upon the location of the onboard ports.

Vectored interrupt 1 (trap address 30H) is genrally used for the real-time clock interrupt service routine. The real-time clock is physically connected on the CPU cardby a jumper from the RTC pad in the upper left hand side of the board to VT1 in the interrupt jumper area.

2.5 SINGLE-STEP LOGIC

The single-step logic hardware uses vectored interrupt 0 for the purpose of executing a single instruction of a program being tested and returning to a fixed location (38H) in RAM. Issuing an output instruction to the single-step port enables the single step logic. This also disables (masks) all other interrupts. The logic will count out two instruction cycles and then generate an interrupt which is vectored to location 38H. The single-step logic is immediately reset and its interrupts that were masked are un-masked. Note that it is still not possible to be interrupted until an enable interrupt (EI) instruction is executed.

The two instructions normally executed after the output instruction are a return (to the program being single-stepped) and one instruction out of the program being single-stepped.

2.6 VECTORED INTERRUPTS

The eight vectored interrupts traps in the 8080A chip are located every eight bytes in the first 64 bytes of memory (see Figure 13).

Interrupt Number	Address (Hex)
0	38*
1	30
2	28
3	20
4	18
5	10
6	8
7	0**

*This location is also used for the single-step trap.
**This location is also used for the power-on reset

Figure 13.

The first trap is also used by the reset function on the CPU, so it is generally not available for use as an interrupt. The last trap (38H) is used by the single-step logic on the CPU card, so it is also not available if the single-step function is to be used. Consequently, only six (or seven) vectored interrupts are generally available. The eight bytes at each trap location are usually not enough for the interrupt service routine, but are enough to save the state of the CPU and jump to the actual routine. The interrupt request lines are labeled VI0 through VI7 on the S-100 bus. To initiate an interrupt, the appropriate line must be brought low and held low until the requesting device has been serviced. The CPU reads the interrupt priority during the interrupt acknowledge cycle (INTA) and jumps to the memory location of the highest priority interrupt trap that is active at that time. VI7 is the highest priority interrupt and VI0 the lowest. If VI1, VI3, and VI6 are all active (low) during INTA, the CPU will jump to location 8. All interrupts are disabled by the CPU after the INTA cycle.

Nested interrupts are not provided, as the interrupt lines are not individually maskable. All interrupts may be enabled or disabled by using the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. The interrupt service routine (ISR) is entered with interrupts disabled, they may be reenabled only after the ISR has been completed and the interrupt request has been reset The manner of reseting the interrupt is dependent upon the particular I/O controller being service but is usually done by issuing an input or output instruction to one of the ports on the controller.

The EI instruction must not be executed until after the interrupt has been reset.

The following program is a sample ISR for the real time clock:

```
        ORG     30H     ;RTC TRAP
        PUSH    PSW     ;SAVE REGISTERS
        PUSH    H       ;USED IN ISR
        JMP     RTCISR  ;JOMP AROUND NEXT
                        ;TRAP LOCATION
        ORG     HIMEM   ;WHERE HIMEM IS AN ADDRESS
                        ;HIGHER IN MEMORY
RTCISR: OUT     8       ;RESET RTC
                        ;INTERRUPT
        LHLD    CTR     ;INCREMENT
        INX     H       ;RTC COUNTER
        SHLD    CTR     ;IN RAM
        POP     H       ;RESTORE REGISTERS
        POP     PSW
        EI              ;RE-ENABLE INTERRUPTS
        RET             ;RETURN TO MAIN
                        ;PROGRAM
```

Note that it is up to the programmer to save and restore the CPU registers used in the ISR.

When using the CPU card with the POLY 88 monitor ROM installed, the interrupt service routines are handled by a table in CPU RAM; see the monitor ROM manual for details.

3. Customer Installed Options

These options are used when particular cards require these signals. They need not be installed to use the CPU card in a POLY 88 or System 88.

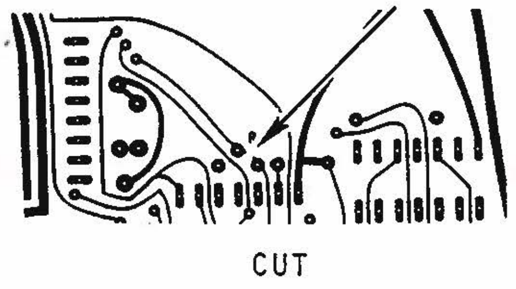
3.1 SINTA

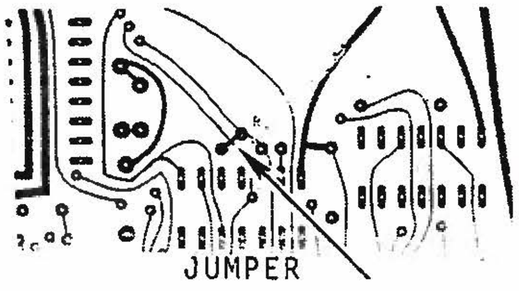
SINTA may be generated by reconfiguring jumpers on the CPU card. Remove the jumper from pad W near IC14 to pad W near IC17. Jumper W and the adjacent pad.

Remove the trace from pad Z near PI-27 to pad Z near PI-48. Jumper the two pads near PI-48 located on either side of the Z. This modification removes the WAIT signal from the S-100 bus.

3.2 PINT

PINT may be enabled by cutting trace R on the back of the CPU card (located near IC20). After cutting this trace jumper the middle and right pads in area R on the top of the card, as shown below:





Note that if PINT is installed, the vectored interrupt facility is disabled and PolyMorphic Systems software will not operate.

3. 3 PHANTOM MEMORY

Memory external to the CPU card is normally quite simple to use. Memory cards simply plug into the S-100 bus after you have selected an appropriate starting address. One exception is the block of memory between addresses 0000 hex and 0FFF hex; the CPU disregards any external memory at this address unless you have made the appropriate modification. The CPU hardware modifications necessary for use of a second block of memory at lower addresses is fairly simple, as it was considered when the card was designed. Some memory cards have "phantom" capabilities on bus pin 67. These offer an advantage when making this modification but are not necessary in many cases. The POLY monitor ROM (if you have it) will be disabled so some software additions will be necessary. The software additions can be lengthy or short depending on your particular application.

As a general rule, make sure the normal functions of the system are operational before attempting any special alterations. Cut the short trace "HH" and remove a small section. This trace is near the upper left corner of the CPU card when viewing the bottom (non-component side). Install a jumper (on the non-component side of the board) between the pad at the right end of the trace "HH" (ungrounded pad) and pad "H" directly above capacitor C9 (left end of card when viewed from top). Pad "H" is labeled only on the component side. Two pads are provided so you can use a jumper to replace trace "HH" if it becomes necessary.

If the memory card you will use as a second block of lower memory has a "phantom" facility at bus pin 67, connect a jumper between bus pin 67 and pin 6 of IC 12, on the non-component side of the CPU board. Bus pin 67 has a jumper pad attached; pin 6 of IC 12 does not have a jumper pad.

Be sure to double check the connections you make before applying power; misplaced jumpers are a common source of serious damage to integrated circuits.

IC 30 is used as a latch to store baud rate, serial device select, and onboard disable status, it will be referred to by the name BRG (baud rate generator latch) for ease of discussion. The BRG output pins continually display the aforementioned status acccording to Figure 14.

To disable the onboard memory, set bit 5 (of bits 0 through 7) HI in the accumulator and output to port 4 (BRG).

```
          Example:  MVI A,20H; SET BIT 5 HI
                    OUT 4    ; LATCH NEW STATUS IN BRG
```

To re-enable the onboard memory set bit 5 of port 4 LO or press the front panel reset button. The latter resets the BRG and executes the monitor which places a given status in the BRG.

The other hardware change suggested for memory cards having a "phantom" facility enables the memory card when pin 7 of baud rate generator (BRG) is HI (onboard memory disable) and disables the memory card when pin 7 is LO (assuming bus pin 67 disables the memory card when HI). The "phantom" facility is not absolutely necessary, but without it, writing into on-board memory (including stack ops performed by the monitor) also writes into the corresponding off-board memory location.

Remember, when the onboard memory is disabled, the POLY monitor ROM is disabled. Thus, you must have any necessary control software (such as keyboard input, video display, and cassette tape reader routines) located in memory external to the CPU board.

3.4 Relocation of CPU Onboard Memory

The onboard memory adddress can be shifted up if you do not wish to use the POLY monitor ROM. It is important to remember the POLY monitor will work only if its starting address is 0000.

There are three possible starting addresses that are easily implemented on the POLY CPU: 0000 (which is preselected on the card), 8000 hex (32K), and E000 hex (56K). If you wish to select a starting location other than 0000, cut trace "J" and remove a small section. Trace "J" is between two closely spaced pads on the non-component side, immediately to the right of IC 35. Install a jumper with sleeving between pad "J" (the one nearest the regulators) and pad "R" for a starting location of 8000 hex; install a jumper between pad "J" (nearest the regulators) and pad "S" for a starting location of E000 hex.

CPU MEMORY ADDRESSING

Address*			Function
J	S	R	
0-3FF	8000-83FF	E000-E3FF	ROM #1
400-7FF	8400-87FF	E400-E7FF	ROM #2
800-BFF	8800-8BFF	E800-EBFF	ROM #3
C00-DFF	8C00-8DFF	EC00-EDFF	Onboard RAM
E00-FFF	8E00-8FFF	EE00-EFFF	Onboard RAM **

**Note that the CPU/8 Memory Addressing chart above is the same RAM as the 512 bytes beneath it. (Bit 9 of the address is not decoded.)

CPU I/O ADDRESSING

Address (Hex)*			R/W	Function
J	S	R		
0,2	80,82	E0,02	R	USART status byte
0,2	80,82	E0,E2	W	USART command/mode
1,3	81,83	E1,E3	R	USART received data
1,3	81,83	E1,E3	W	USART transmit data
4-7	84-87	E4-E7	R	Unused
4-7	84-87	E4-E7	W	Baud Rate Latch
8-B	88-8B	E8-EB	R	Unused
8-B	88-8B	E8-EB	W	Reset real-time clock
C-F	8C-8F	EC-EF	R	Unused
C-F	8C-8F	EC-EF	W	Start single step

* Note that jumpers J, S and R determine the base address of the CPU memory and I/O address space.

FIGURE 14: TABLE OF CPU ADDRESSES

3. 5 REAL-TIME CLOCK JUMPER

The RTC jumper is installed on the back of the CPU card between pad VT1 (adjacent and connected to the bottom leg of R5 near IC2) and pad A near IC20. This jumper is installed at the factory on factory-assembled cards and must be present if you are to use PolyMorphic Systems software. If the real-time clock is not to be used, the jumper may be removed.

4. PolyMorphic Systems 8080 CPU Card Bus

Front-Panel Control Signals

A number of signals generated or used by the Altair2/IMSAI front panels are not used by the POLY system. These include 21(UNPROT), 22(SS), 53(SSWI-), 54(EXTCLR), 56(DTDTB), 57(FRDY), 69(PS-), 70 (PROT), and 71(RUN).

Status Signals

Two status signals defined in the Altair/IMSAI bus are not present in the POLY. The SSTACK (98) signal is not generated and the

PINTE (pin 28) signal is not brought out to the bus but is available as a test point on the CPU card. Other status lines are optional (see Customer Installed Options).

Pin Number	Label	Description
1	+8V	Unregulated voltage, regulated to +5V on card
2	+16V	Unregulated voltage, regulated to +12V on card
3	XRDY+	External ready--ready input to CPU
4	VI0-	Vectored interrupt line
5	VI1-	Vectored interrupt line
6	VI2-	Vectored interrupt line
7	VI3-	Vectored interrupt line
8	VI4-	Vectored interrupt line
9	VI5-	Vectored interrupt line
10	VI6-	Vectored interrupt line
11	VI7-	Vectored interrupt line
12	--	Unused
13	--	Reserved for bus control
14	--	Reserved for bus control
15	--	Reserved for bus control
16	--	Reserved for bus control
17	--	Reserved for bus control
18	--	Reserved for bus control
19	--	Reserved for bus control
20	--	Unused
21	--	Unused
22	--	Reserved for bus control
23	--	Reserved for bus control
24	02+	Phase 2 clock from CPU
25	01+	Phase 1 clock from CPU
26	--	Reserved for bus control
27	WAIT+	CPU in wait cycle
28	--	Unused
29	A5+	Address line
30	A4+	Address line
31	A3+	Address line
32	A15+	Address line
33	A12+	Address line
34	A9+	Address line
35	DO1+	Data out line
36	DO0+	Data out line
37	A10+	Address line
38	DO4+	Data out line
39	DO5+	Data out line
40	DO6+	Data out line
41	DI2+	Data in line
42	DI3+	Data in line
43	DI7+	Data in line
44	SM1+	Instruction fetch cycle
45	SOUT+	Output -- during this machine cycle data is transferred from the CPU to an output port.
46	SINP+	Input -- during this machine cycle data is transferred from an input port to CPU.
47	SMEMR+	Memory read cycle
48	SHLTA+	Halt -- CPU has entered a halt state

49	CLOCK-	Phase 2 clock
50	GND	System ground
51	+8V	Unregulated voltage, regulated on card to +5V
52	-16V	Unregulated voltage, regulated on card to -12V
53	--	Unused
54	--	Unused
55	RTC+	Real-time clock--TTL 50/60Hz clock or half wave rectified 50/60 Hz signal
56	--	Unused
57	--	Unused
58	--	Unused
59	--	Reserved
60	--	Reserved
61	--	Reserved
62	--	Reserved
63	--	Reserved for MP/DMA controller
64	--	Reserved for MP/DMA controller
65	--	Reserved for MP/DMA controller
67	PHANTOM-	Used to disable RAM addressing*
68	MWRITE+	Memory--write strobe for memory cycle
69	--	Unused
70	--	Unused
71	--	Unused
72	PRDY+	Processor ready--ready input to CPU
73	PINT-	Processor Interrupt--used only with external interrupt controller
74	--	Reserved for bus control
75	PRESET-	Reset--from front panel pushbutton--sets PC to 0
76	PSYNC+	Sync--identifies beginning of machine cycle
77	PWR-	Write--CPU write strobe
78	PDBIN+	Data in--CPU read strobe
79	A0+	Address line
80	A1+	Address line
81	A2+	Address line
82	A6+	Address line
83	A7+	Address line
84	A8+	Address line
85	A13+	Address line
86	A14+	Address line
87	A11+	Address line
88	DO2+	Data out line
89	DO3+	Data out line
90	DO7+	Data out line
91	DI4+	Data in line
92	DI5+	Data in line
93	DI6+	Data in line
94	DI1+	Data in line
95	DI0+	Data in line
96	SINTA+	Interrupt acknowledge cycle*
97	--	Unused
98	--	Unused
99	POC-	Power on clear--generated by PRESET+ or power turn on used to reset CPU and I/O devices
100	GND	System ground

* jumper option

BUS DC SPECIFICATIONS

All outputs (except HLTA, POC):

Logic 0:	0.5V max. at 48mA
Logic 1:	2.4V min. at -5.2mA

HLTA, POC:

Logic 0:	0.45V max. at 40mA
Logic 1:	1000 ohms pullup to +5V

All inputs (except VI0-7, XRDY, PRDY, PRESET):

Logic 0:	0 to 0.8V, 400 A max.
Logic 1:	2.0 to 4.75V, 20 A max.

VI0-7:

Logic 0:	0 to 0.8V, 1.6mA max.
Logic 1:	2.0 to 4.75V, 400 A max.

XRDY, PRDY, PRESET:

Logic 0:	0 to 0.8V, 2.5 mA
Logic 1:	1.9 to 4.75 (open circuit, 2200 ohms pullup to +5V)

SERIAL I/O PORT CONNECTORS

Pin	Name	Function
1	TXD	Transmitter Data (Output)
2	RXD	Receiver Data (Input)
3	RTS	Request To Send (Output)
4	CTS	Clear To Send (Input)
5	DTR	Data Terminal Ready (Output)
6	DSR	Data Set Ready (Input)*
7	TXC	Transmitter Clock (Output)
8	RXC	Receiver Clock; (Input)
9	PS	Port Select (0 or 1) (Output)
10	GND	Ground (Output)
11	-5V	Power Supply (Output)
12	-12V	Power Supply (Output)
13	+12V	Power Supply (Output)
14	+5V	Power Supply (Output)

*Used for speed select output (0=Byte, 1=PolyPhase) on cassette port (#2).

Each connector is a standard 14-pin DIP socket.

Output Drive:

Source:	400 A at +2.4V
Sink:	8mA at +0.5V
Logic 1:	+2 to +4.15V

Logic 0:	0 to 0.8V

Input Loading:

Source:	< + or -100 A
Sink:	< + or -100 A
Logic 0:	-0.5 to 0.8V
Logic 1:	+2.0 to 4.75V

Clock Frequency (RXC, TXC): 0 to 56,000 Hz (at 50% duty cycle)

NRZ Data Rate (TXD, RXD): 0 to 56,000 baud

Clock Phasing (Synchronous): Positive clock transition is middle of bit cell

Power Supplies:

+5V + or -5% at 100mA
+12V + or -5% at 50mA
-12V + or -5% at 25mA
-5V + or -5% at 50mA

REFERENCE LIST: FUNCTIONAL LISTING OF THE 8080 INSTRUCTION SET

MOVE				ACCUMULATOR *			
40	MOV B,B	60	MOV H,B	80	ADD B	A0	ANA B
41	MOV B,C	61	MOV H,C	81	ADD C	A1	ANA C
42	MOV B,D	62	MOV H,D	82	ADD D	A2	ANA D
43	MOV B,E	63	MOV H,E	83	ADD E	A3	ANA E
44	MOV B,H	64	MOV H,H	84	ADD H	A4	ANA H
45	MOV B,L	65	MOV H,L	85	ADD L	A5	ANA L
46	MOV B,M	66	MOV H,M	86	ADD M	A6	ANA M
47	MOV B,A	67	MOV H,A	87	ADD A	A7	ANA A
48	MOV C,B	68	MOV L,B	88	ADC B	A8	XRA B
49	MOV C,C	69	MOV L,C	89	ADC C	A9	XRA C
4A	MOV C,D	6A	MOV L,D	8A	ADC D	AA	XRA D
4B	MOV C,E	6B	MOV L,E	8B	ADC E	AB	XRA E
4C	MOV C,H	6C	MOV L,H	8C	ADC H	AC	XRA H
4D	MOV C,L	6D	MOV L,L	8D	ADC L	AD	XRA L
4E	MOV C,M	6E	MOV L,M	8E	ADC M	AE	XRA M
4F	MOV C,A	6F	MOV L,A	8F	ADC A	AF	XRA A
50	MOV D,B	70	MOV M,B	90	SUB B	B0	ORA B
51	MOV D,C	71	MOV M,C	91	SUB C	B1	ORA C
52	MOV D,D	72	MOV M,D	92	SUB D	B2	ORA D
53	MOV D,E	73	MOV M,E	93	SUB E	B3	ORA E
54	MOV D,H	74	MOV M,H	94	SUB H	B4	ORA H
55	MOV D,L	75	MOV M,L	95	SUB L	B5	ORA L
56	MOV D,M		--------	96	SUB M	B6	ORA M
57	MOV D,A	77	MOV M,A	97	SUB A	B7	ORA A
58	MOV E,B	78	MOV A,B	98	SBB B	B8	CMP B
59	MOV E,C	79	MOV A,C	99	SBB C	B9	CMP C
5A	MOV E,D	7A	MOV A,D	9A	SBB D	BA	CMP D
5B	MOV E,E	7B	MOV A,E	9B	SBB E	BB	CMP E
5C	MOV E,H	7C	MOV A,H	9C	SBB H	BC	CMP H
5D	MOV E,L	7D	MOV A,L	9D	SBB L	BD	CMP L
5E	MOV E,M	7E	MOV A,M	9E	SBB M	BE	CMP M
5F	MOV E,A	7F	MOV A,A	9F	SBB A	BF	CMP A

* = All flags (C,Z,S,P,AC) affected.

RETURN

C9	RET
CØ	RNZ
C8	RZ
DØ	RNC
D8	RC
EØ	RPO
E8	RPE
FØ	RP
F8	RM

LOAD/STORE

ØA	LDAX B
1A	LDAX D
2A	LHLD Adr
3A	LDA Adr
Ø2	STAX B
12	STAX D
22	SHLD Adr
32	STA Adr

RESTART

C7	RST Ø
CF	RST 1
D7	RST 2
DF	RST 3
E7	RST 4
EF	RST 5
F7	RST 6
FF	RST 7

ACC IMMEDIATE *

C6	ADI d8
CE	ACI d8
D6	SUI d8
DE	SBI d8
E6	ANI d8
EE	XRI d8
F6	ORI d8

LOAD IMMEDIATE

Ø1	LXI B,d16
11	LXI D,d16
21	LXI H,d16
31	LXI SP,d16

INPUT/OUTPUT

D3	OUT d8
DB	IN d8

SPECIALS

EB	XCHG
27	DAA *
2F	CMA
37	STC **
3F	CMC **

INCREMENT a

Ø4	INR B
ØC	INR C
14	INR D
1C	INR E
24	INR H
2C	INR L
34	INR M
3C	INR A
Ø3	INX B
13	INX D
23	INX H
33	INX SP

STACK OPS

C5	PUSH B
D5	PUSH D
E5	PUSH H
F5	PUSH PSW
C1	POP B
D1	POP D
E1	POP H
F1	POP PSW *
E3	XTHL
F9	SPHL

DECREMENT a

Ø5	DCR B
ØD	DCR C
15	DCR D
1D	DCR E
25	DCR H
2D	DCR L
35	DCR M
3D	DCR A
ØB	DCX B
1B	DCX D
2B	DCX H
3B	DCX SP

Adr =	16 bit address
* =	All flags (C,Z,S,P,AC) affected.
** =	Only CARRY flag affected.
a =	All flags except CARRY affected (exception: INX & DCX affect no flags).
d8 =	Constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity.
d16 =	Constant, or logical/arithmetic expression that evaluates to a 16-bit data quantity.

JUMP

C3	JMP Adr
C2	JNZ Adr
CA	JZ Adr
D2	JNC Adr
DA	JC Adr
E2	JPO Adr
EA	JPE Adr
F2	JP Adr
FA	JM Adr
E9	PCHL

MOVE IMMEDIATE

Ø6	MVI B,d8
ØE	MVI C,d8
16	MVI D,d8
1E	MVI E,d8
26	MVI H,d8
2E	MVI L,d8
36	MVI M,d8
3E	MVI A,d8

CALL

CD	CALL Adr
C4	CNZ Adr
CC	CZ Adr
D4	CNC Adr
DC	CC Adr
E4	CPO Adr
EC	CPE Adr
F4	CP Adr
FC	CM Adr

DOUBLE ADD **

Ø9	DAD B
19	DAD D
29	DAD H
39	DAD SP

CONTROL

ØØ	NOP
76	HLT
F3	DI
FB	EI

ROTATE **

Ø7	RLC
ØF	RRC
17	RAL
1F	RAR

FLAG BYTE STACK FORMAT

7	6	5	4	3	2	1	Ø
S	Z	Ø	AC	Ø	P	1	C

Adr = 16 bit address

d8 = Constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity.

** = Only CARRY flag affected.

REFERENCE LIST: NUMERICAL LISTING OF THE 8080 INSTRUCTION SET

00	NOP	20	---	40	MOV B,B	60	MOV H,B
01	LXI B,d16	21	LXI H,d16	41	MOV B,C	61	MOV H,C
02	STAX B	22	SHLD Adr	42	MOV B,D	62	MOV H,D
03	INX B	23	INX H	43	MOV B,E	63	MOV H,E
04	INR B	24	INR H	44	MOV B,H	64	MOV H,H
05	DCR B	25	DCR H	45	MOV B,L	65	MOV H,L
06	MVI B,d8	26	MVI H,d8	46	MOV B,M	66	MOV H,M
07	RLC	27	DAA	47	MOV B,A	67	MOV H,A
08	---	28	---	48	MOV C,B	68	MOV L,B
09	DAD B	29	DAD H	49	MOV C,C	69	MOV L,C
0A	LDAX B	2A	LHLD Adr	4A	MOV C,D	6A	MOV L,D
0B	DCX B	2B	DCX H	4B	MOV C,E	6B	MOV L,E
0C	INR C	2C	INR L	4C	MOV C,H	6C	MOV L,H
0D	DCR C	2D	DCR L	4D	MOV C,L	6D	MOV L,L
0E	MVI C,d8	2E	MVI L,d8	4E	MOV C,M	6E	MOV L,M
0F	RRC	2F	CMA	4F	MOV C,A	6F	MOV L,A
10	---	30	---	50	MOV D,B	70	MOV M,B
11	LXI D,d16	31	LXI SP,d16	51	MOV D,C	71	MOV M,C
12	STAX D	32	STA Adr	52	MOV D,D	72	MOV M,D
13	INX D	33	INX SP	53	MOV D,E	73	MOV M,E
14	INR D	34	INR M	54	MOV D,H	74	MOV M,H
15	DCR D	35	DCR M	55	MOV D,L	75	MOV M,L
16	MVI D,d8	36	MVI M,d8	56	MOV D,M	76	HLT
17	RAL	37	STC	57	MOV D,A	77	MOV M,A
18	---	38	---	58	MOV E,B	78	MOV A,B
19	DAD D	39	DAD SP	59	MOV E,C	79	MOV A,C
1A	LDAX D	3A	LDA Adr	5A	MOV E,D	7A	MOV A,D
1B	DCX D	3B	DCX SP	5B	MOV E,E	7B	MOV A,E
1C	INR E	3C	INR A	5C	MOV E,H	7C	MOV A,H
1D	DCR E	3D	DCR A	5D	MOV E,L	7D	MOV A,L
1E	MVI E,d8	3E	MVI A,d8	5E	MOV E,M	7E	MOV A,M
1F	RAR	3F	CMC	5F	MOV E,A	7F	MOV A,A

d16	=	Constant, or logical/arithmetic expression that evaluates to a 16-bit data quantity.
d8	=	Constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity.
Adr	=	16 bit address