

Chapter 2

THE GENERAL INSTRUMENT CP1600

The CP1600 and the TMS 9900 were the first two NMOS16-bit microprocessors commercially available. Even a superficial inspection of the CP1600 shows it to be more powerful than the National Semiconductor PACE (Or 8900), **yet the CP1600 is not widely used.** This is because General Instrument does not support the CP1600 to the extent that National Semiconductor originally supported PACE, or most manufacturers support their 8-bit microprocessors.

General Instrument's marketing philosophy has been to seek out very high volume customers: General Instrument supports low-volume customers only to the extent that this support would not require substantial investment on the part of General Instrument.

From the viewpoint of the low-volume microprocessor user. General Instrument's marketing philosophy is unfortunate. The CP1600 is an ideal microprocessor for the more sophisticated video games that are appearing, and its rich Instruction set and capable architecture make it an ideal choice for data processing terminals and home computer systems. However, due to its limited support, potential low-volume CP1600 customers are likely to choose another equally capable product.

Three CP1600 parts are available, differentiated only by the clock speeds for which they have been designed.

The CP1600 requires a 3.3 MHz. two-phase clock and generates a 600 nanosecond machine cycle time.

The CP1600 requires a 4 MHz. two-phase clock and generates a 500 nanosecond machine cycle time.

The CP1610 requires a 2 MHz. two-phase clock and generates a 1 microsecond cycle time.

In addition to the CP1600 microprocessors themselves, the CP1680 Input/Output Buffer (IOB) is described in this chapter. Additional support devices for the CP1600 may be found in An Introduction to Microcomputers: Volume 3 – Some Real Support Devices.

The sole source for the CP1600 is:

GENERAL INSTRUMENT
Microelectronics Division
600 West John Street
Hicksville, New York 11802

There is no second source for the CP1600. General Instrument has a policy of discouraging second sources for its product line.

The CP1600 is fabricated using NMOS ion implant LSI technology; the device is packaged as a 40-pin DIP. Three power supplies are required: +12V, +5V, and -3V.

THE CP1600 MICROCOMPUTER SYSTEM OVERVIEW

Logic of our general microcomputer system which has been implemented by the CP1600 CPU is illustrated in Figure 2-1.

Observe that the CP1600 requires external logic to create its various timing and clock signals. Some bus interface logic is shown as absent because a number of devices must surround the CP1600: these include:

- 1) An address buffer, since data and addresses are multiplexed on a single 16-bit bus.
- 2) Buffer amplifiers to provide the power required by the type of memory and I/O devices that will normally be connected to a CP1600 CPU.
- 3) A one-of-eight decoder chip to create eight individual control signals out of three controls output by the CP1600.
- 4) A one-of-sixteen multiplex chip to funnel sixteen external status signals into the CP1600 if using external branches.

Were you to compare Figure 2-1 with an equivalent figure for a low-end microprocessor such as the SC/MP (which is described in Chapter 3 of the Osborne 4 & 8-Bit Microprocessor Handbook (Osborne/McGraw-Hill, 1980), the CP1600 might appear to offer fewer logic functions: but within the functions it does provide the CP1600 provides considerably more logic and program execution capabilities. Where low-end microprocessors choose to condense onto a single chip, simple implementations of different logic functions, high-end products such as the CP1600 choose to provide more devices greater capabilities on each device.

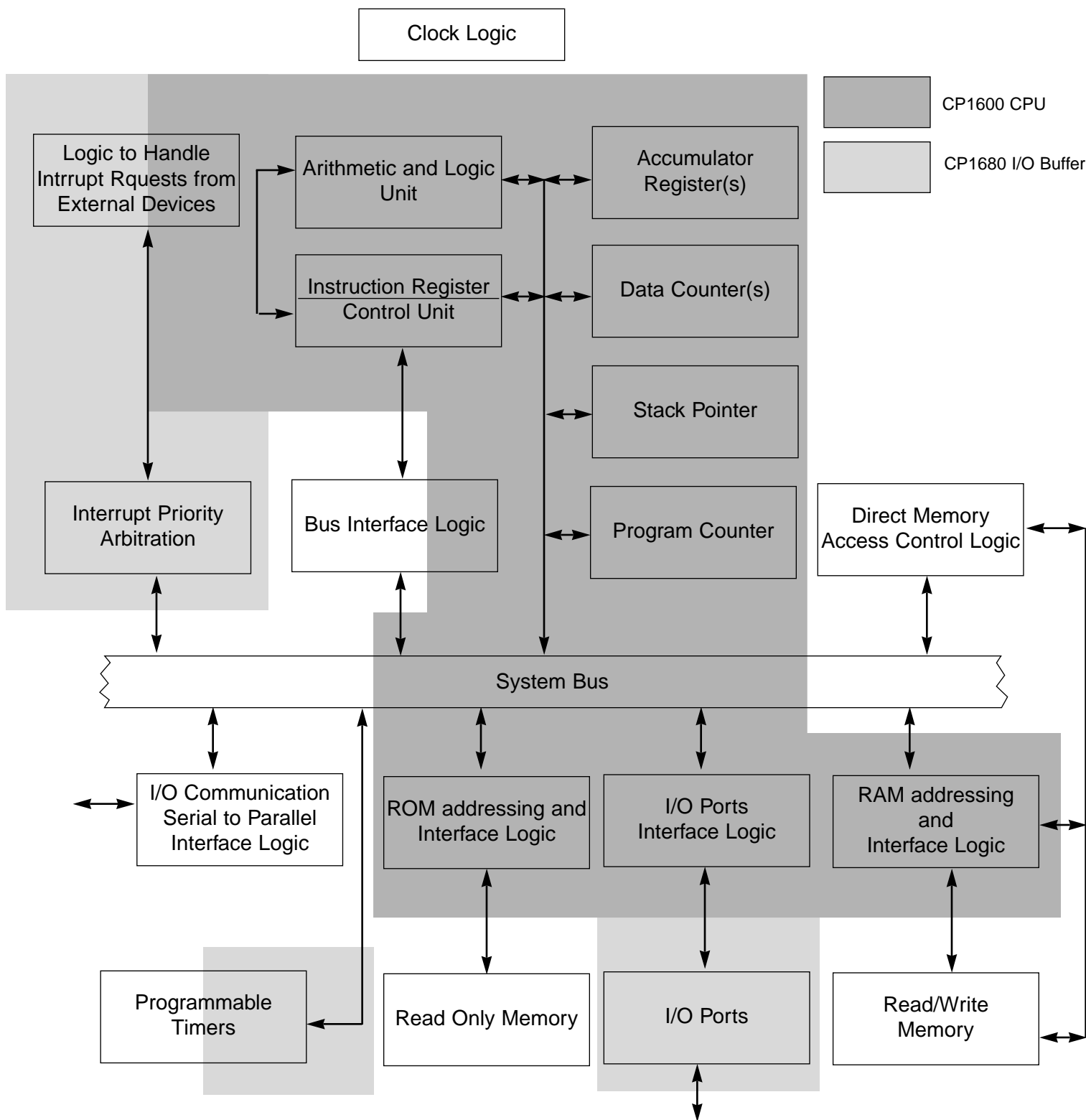
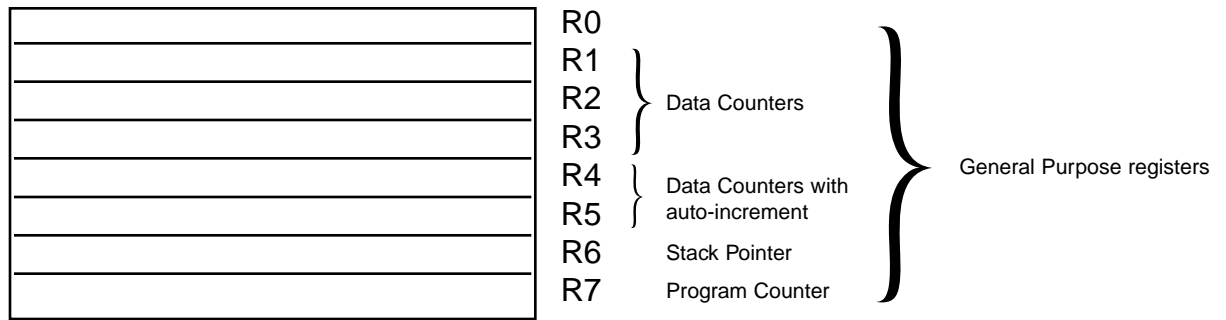


Figure 2-1. Logic of the CP1600 CPU and CP1680 I/O Buffer

CP1600 PROGRAMMABLE REGISTERS

The CP1600 has eight 16-bit programmable registers, which may be illustrated as follows:



The way in which the registers illustrated above are used is unusual when compared to other microcomputers described in this book. All eight 16-bit registers can be addressed as though they were general purpose registers: however, only Register R0 has no other assigned function. We may therefore look upon Register R0 as the Primary Accumulator for this CPU.

Registers R1, R2, and R3 serve as general purpose registers, But may also be used as Data Counters.

In addition to serving as general purpose registers, R4 and R5 may be used as auto-incrementing Data Counters. Memory reference instructions that identify Register R4 or R5 as holding the implied memory address will cause the contents of Register R4 or R5 to be incremented — after the memory reference instructions have completed execution.

Registers R6 and R7, in addition to being accessible as general purpose registers, also serve as a Stack Pointer and a Program Counter, respectively.

Having the Stack Pointer accessible as a general purpose register makes it quite simple to maintain more than one Stack in external memory; also, you can easily address the Stack as data memory using the Stack Pointer as a Data Counter.

Having the Program Counter accessible as a general purpose register can be useful when executing various types of conditional branch logic.

While having the Stack Pointer and the Program Counter accessible as though they were general purpose registers may appear strange, this is a feature of the PDP-11 minicomputer — and is a very powerful programming tool.

CP1600 MEMORY ADDRESSING MODE

The CP1600 addresses memory and I/O devices within a single address space.

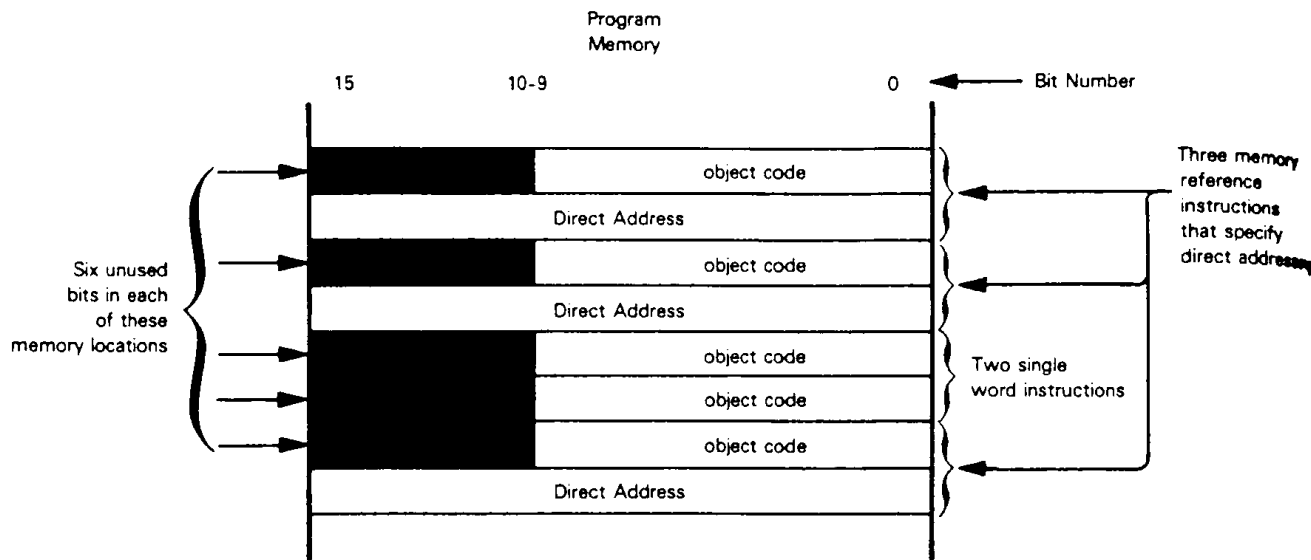
When referencing external memory, you can use direct addressing, implied addressing, or implied addressing with auto-increment.

Direct addressing instructions are all two or more words long, where the second or last word of the instruction object code provides a 16-bit direct address.

CP1600 DIRECT ADDRESSING

CP1600 direct addressing instructions are complicated by the fact that CP1600 program memory is frequently only 10 bits wide. That is to say, even though the CP1600 is a 16-bit microprocessor, its instruction object codes are only 10 bits wide. If program memory is only 10 bits wide, then direct addresses will only be 10 bits wide. A 10-bit direct address will access the first 1024 words of memory only.

Were you to implement a 16-bit wide program memory, then you could directly address up to 65,536 words of memory, however, six bits of the first object program word for every instruction in program memory would be wasted. This may be illustrated as follows:

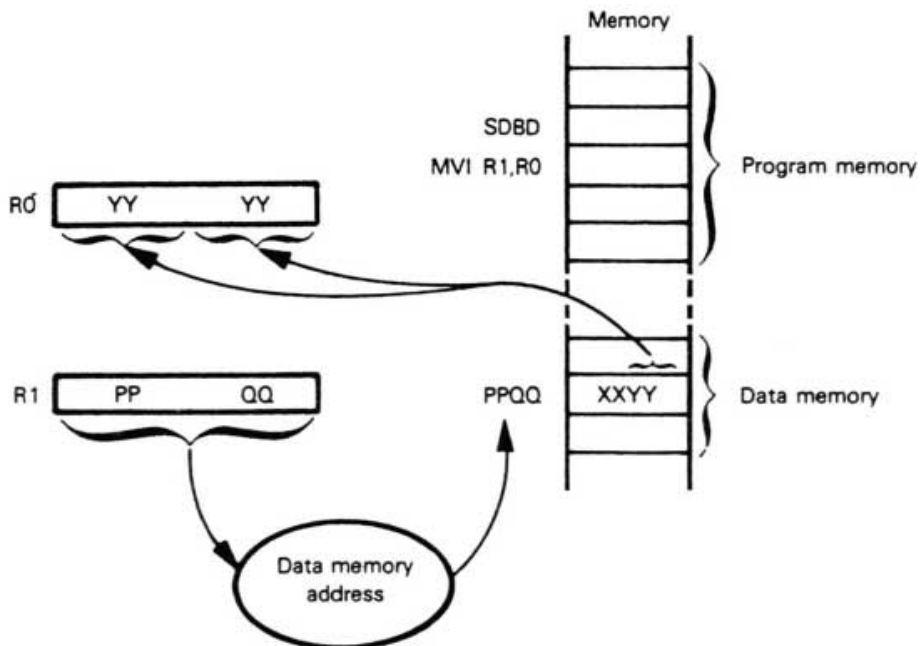


Instructions that reference memory using implied addressing identify general purpose Register R1, R2, or R3 as containing the implied address.

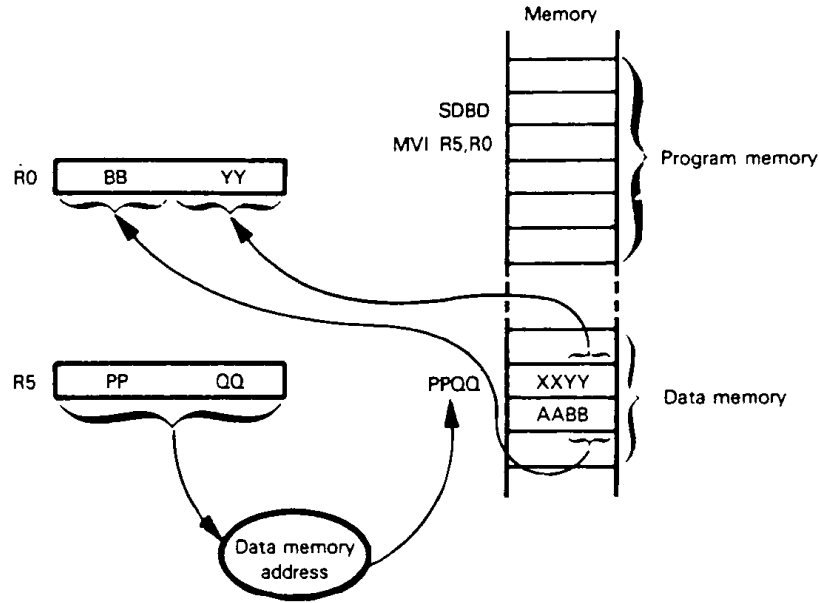
**CP1600
IMPLIED
ADDRESSING**

A memory reference instruction which identifies Register R4 or R5 as providing the external memory address will always cause Register R4 or R5 contents to be incremented following the memory access; thus you have implied memory addressing with auto-increment.

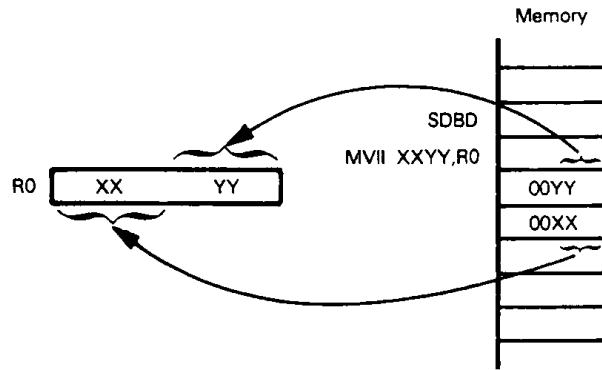
Memory reference instruction, that specify implied memory addressing via Register 1, 2, 3, 4, or 5 can access 8-bit memory. An SDBD instruction executed directly before a valid memory reference instruction forces the reference instruction to access memory one byte at a time if implied memory addressing via Register 1, 2, or 3 is specified, then the same byte of memory will be accessed twice For an instruction that loads the contents of data memory into Register R0, this may be illustrated as follows:



If Registers R4 or R5 provides the implied memory address for the instruction which follows an SDBD instruction, then the implied memory address is incremented twice, and two sequential low-order bytes of data are accessed. For an Instruction which loads data into Register R0, this may be illustrated as follows:



The SDBD instruction may also precede an immediate Instruction. Now the immediate data will be fetched from the low-order byte of the next two sequential program memory locations This may be illustrated as follows



Without the preceding SDBD instruction, an immediate instruction will access the next single program memory word to find the required immediate data. Ten or more bits of immediate data will be accessed depending on the width of program memory words.

The CP1600 has no Stack reference instructions such as a Push or Pull; rather, a variety of memory reference instructions can identify Register R6 as providing the implied address. When Register R6 provides the implied address, it is treated as an upward migrating Stack Pointer. When a memory write operation specifies Register R6 as providing the implied memory address, Register R6 contents will be incremented following the memory write. A memory read instruction that specifies Register R6 as providing the implied memory address will cause the contents of Register R6 to be decremented before the read operation occurs.

CP1600 STACK ADDRESSING

An unusual feature of the CP1600 is the fact that a variety of secondary memory reference instructions can also be reference memory via the Stack Pointer. When these instructions are executed. Register R6 contents are decremented before the memory access occurs — as though a Pull operation from the Stack were being executed.

Logically, Register R6, the Stack Pointer, is being handled as though it were a Data Counter with post-increment and pre-decrement.

Jump instructions use direct memory addressing. Jump instructions are all three words long. The direct address is computed from the second and third memory words as follows:



AAAAAABBBBBBBBBB Jump address (binary)
 YY are enable/disable bits for interrupts
 XX identity the register where the return address will be stored for JSR
 XX and YY are described in detail in Table 2-4.

You can enable or disable interrupts whenever you execute a Jump or Jump-to-Subroutine instruction.

The only difference between a Jump instruction and a Jump-to-Subroutine instruction is that the Jump-to-Subroutine instruction saves the Program Counter contents in Register 4, 5, or 6. The two high-order bits (XX) or the second Jump-to-Subroutine object code word specifies which of the three registers will be used to hold the return address.

Jump-to-Subroutine instructions, like the Jump instruction, allow direct memory addressing only.

CP1600 STATUS AND CONTROL FLAGS

The CP1600 CPU has four of the standard status flags; in addition, it has some unusual control signals.

These are the four standard status flags:

Sign (S). This status is set equal to the high-order bit of any arithmetic operation result.

Zero (Z). This status is set to 1 when any instruction's execution creates a zero result. The status is set to 0 for a nonzero result.

The Carry (C) and Overflow (O) statuses are standard carry and overflow, as described in Volume 1.

Four control signals (EBCA0 - EVCA3) are output during a Branch-on-External (BEXT) instruction. These four signals are output to reflect the low-order four bits of the BEXT instruction's object code. External logic receives these four signals and (depending on their state), may or may not return a high input via EBCI. If EBCI is returned high, then the BEXT instruction will perform a branch; if EBCI is returned low, then the BEXT instruction will cause the next sequential instruction to be executed. The four control signals EBCA0 - EBCA3 therefore provide the CP1600 with a means of testing 16 external conditions.

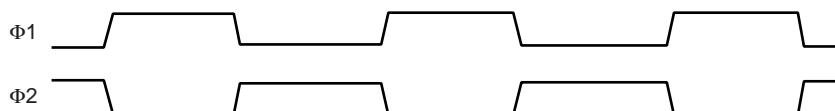
CP1600 CPU PINS AND SIGNALS

CP1600 CPU pins and signals are illustrated in Figure 2-2.

D0 - D15 is a multiplexed Address and Data Bus. Given a total of 40 pins in a package, CP1600 designers have been forced to share 16 pins between addresses and data. **Three control signals BDIR, BC1, and BC2, identify the traffic on the Address/Data Bus. External logic (one MSI chip) must decode these three signals to create eight control signals, as summarized in Table 2-1.**

Remaining signals may be divided into four groups: timing, status/control, interrupt, and DMA.

Two timing clock signals are required: $\Phi 1$ and $\Phi 2$. These are complementary clock signals which may be illustrated as follows:



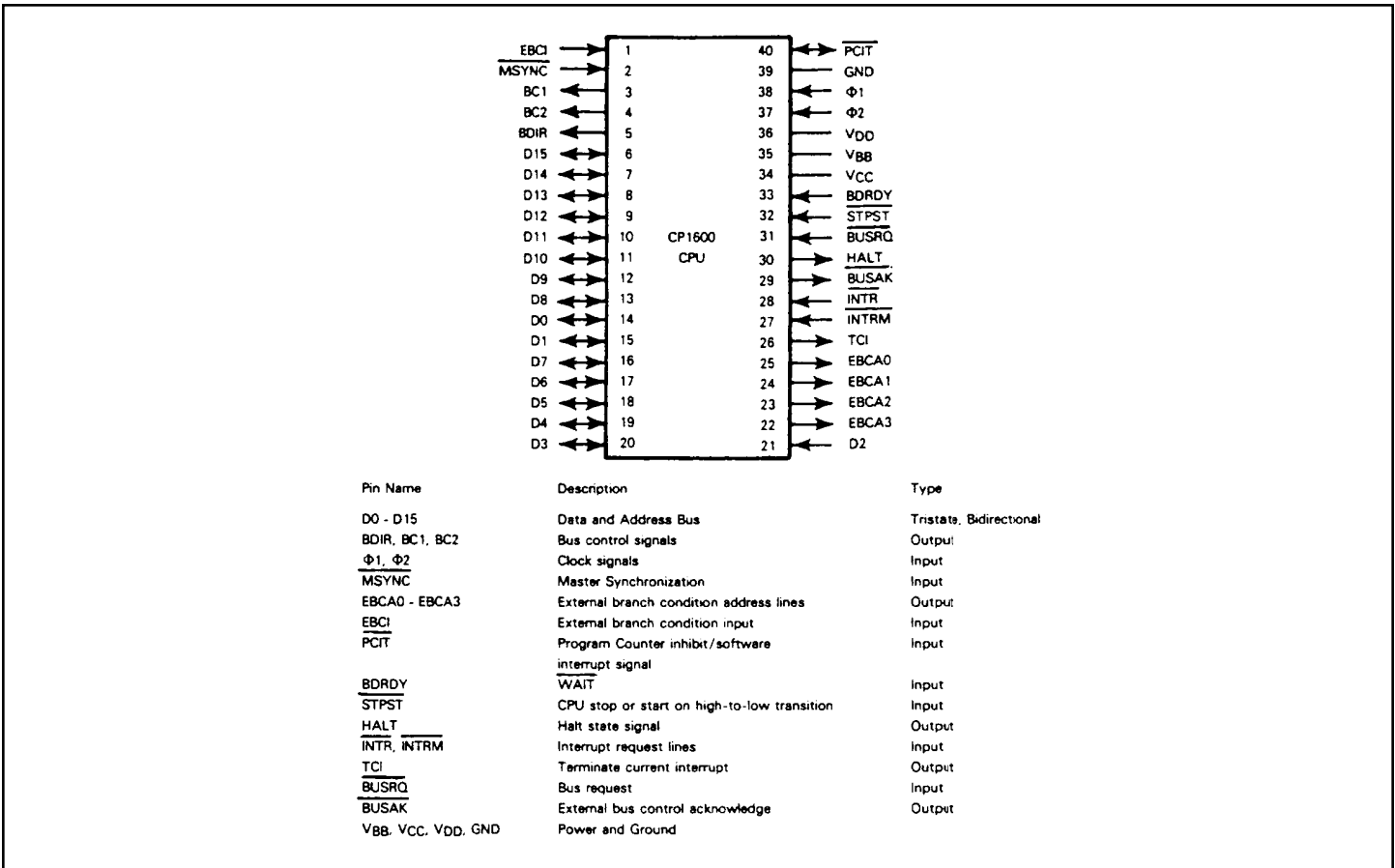


Figure 2-2 CP1600 CPU Signals and Pin Assignments

MSYNC is a somewhat unusual signal, as compared to other microcomputer clock signals in this book. Following powerup, MSYNC must be held low for at least 10 milliseconds. On the subsequent rising edge of MSYNC, logic internal to the CP1600 CPU will synchronize the Φ1 and Φ2 clock signals to start a new machine cycle. Most of the CPU devices we have described in this book use a reset signal or have internal powerup logic which performs this clock synchronization.

Now consider the status and control signals.

First of all, there are the four control outputs which we have already described: **EBCA0 - EBCA3**. There is one conditional Branch instruction (**BEXT**) which will only branch if a high signal is input via **EBCI**. When the BEXT instruction is executed, the low-order four BEXT instruction object code bits are output via EBCA0 - EBCA3. External logic is supposed to decode these four signals by whatever means are appropriate — and thence determine whether EBCI should be input high or low. A high input, as we have just stated, will result in a branch; a low input will cause the next sequential instruction to be executed.

In reality, there is no connection within CP1600 CPU logic between the EBCI input and the four EBCA0-EBCA3 outputs. So far as external logic is concerned, the execution of a BEXT instruction is identified by signal levels output and maintained on the EBCA0 - EBCA3 outputs. While the EBCI input determines whether a branch will or will not occur. How external logic chooses to determine whether EBCI will be set high or low is entirely up to external logic. The only vital function served by EBCA0 - EBCA3 is to identify the instant at which a BEXT instruction is executed.

Another unusual control signal provided by the CP1600 is **PCIT**: this is a bidirectional signal. When input low, this signal prevents the Program Counter from being incremented following an instruction fetch. This signal is also output as a low pulse following execution of a software interrupt instruction. Instruction timing separates the active input and

active output of this signal; providing external logic adheres to timing requirements, a conflict between input and output logic will never arise.

BDRDY is equivalent to the WAIT signal we have described for a number of other microcomputers. BDRDY is input low by any external logic which requires more time in order to respond to an I/O access. Recall that the CP1600 uses a single address space to reference memory or I/O devices. The BDRDY signal causes the CPU to enter a Wait state for as long as BDRDY is being input low; however, during the Wait state, CPU logic is not refreshed. Thus a Wait state cannot last for more than 40 microseconds, or the contents of internal CPU locations will be lost.

STPST a Halt/Reset input, is an edge-triggered signal. When external logic inputs a high-to-low transition via STPST the CPU will complete execution of any interrupt instruction, then will enter a Halt state and output HALT high. If a non-interruptible instruction is being executed, then the Halt state will not begin until completion of next interruptable instruction's execution. The Halt state will last until external logic inputs another high-to-low STPST transition, at which time the Halt output will be returned low and normal programming execution will continue. Execution of the HLT instruction also causes the CP1600 to enter a Halt state as described above.

Let us now look at interrupt signals.

The CP1600 has two interrupt request inputs — **INTR** and **INTRM**. INTR has higher priority than INTRM. INTR cannot be disabled. Typically, INTR will be used to trigger an interrupt upon power failure or other catastrophes.

The interrupt acknowledge signal is created by external logic which must decode the BC1, BC2, and BDIR signals, as shown in Table 2-1. Observe that there are, in fact, two interrupt acknowledge signals: the first (INTAK) acknowledges the interrupt itself, while the second (DAB) is used as a strobe for external logic to return an interrupt address vector. The interrupt sequence is described later in this chapter.

The CP1600 has two additional interrupt related signals which are unusual compared to other microcomputers described in this book.

TCI is output high when an End-of-Interrupt Instruction is executed. This signal makes it easy for external logic to generate interrupt priorities which extend across the execution of an interrupt service routine.

Table 2-1, CP1600 Bus Control Signals

BC1	BC2	BDIR	SIGNAL	FUNCTION
0	0	0	NACT	The CPU is inactive and the Data/Address Bus is in a high impedance state.
0	0	1	BAR	A memory address must be input to the CPU via the Data/Address Bus.
0	1	0	IAB	Acknowledged external interrupt requesting logic must place the starting address for the interrupt service routine on the Address Bus.
0	1	1	DWS	Data write strobe for external memory.
1	0	0	ADAR	This signal identifies a time interval during which the Data/Address Bus is floated, while data input on the Data Bus is being interpreted as the effective memory address during a direct memory addressing operation.
1	0	1	DW	The CPU is writing data into external memory. DW will precede DWS by one machine cycle.
1	0	0	DTB	This is a read strobe which external memory or I/O logic can use in order to place data on the Data/Address Bus.
1	1	1	INTAK	This is an interrupt acknowledge signal. It is followed by IAD which is a strobe telling the external logic which is being acknowledged to identify itself by placing an address vector on the Data/Address Bus.

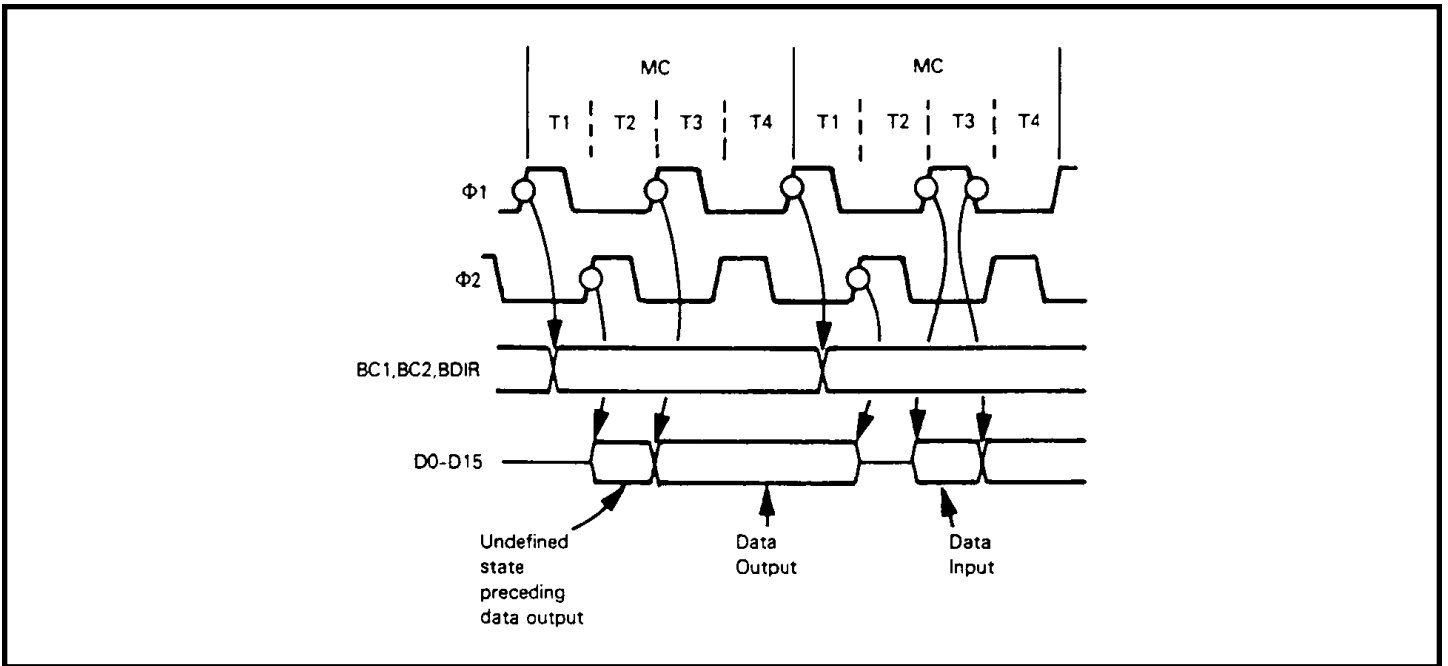


Figure 2-3. CP1600 Machine Cycles and Bus Timing

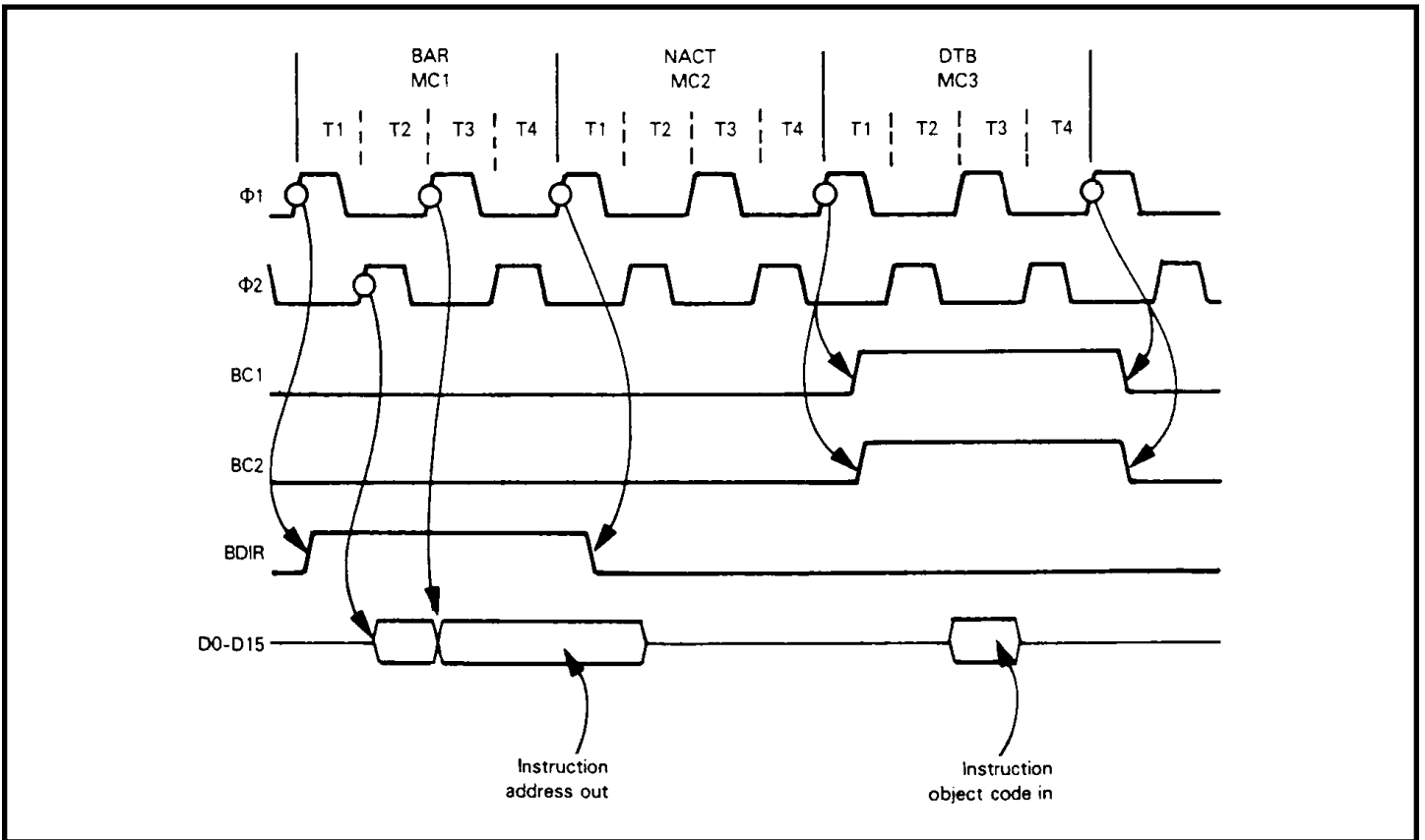


Figure 2-4. CP1600 Instruction Fetch Timing

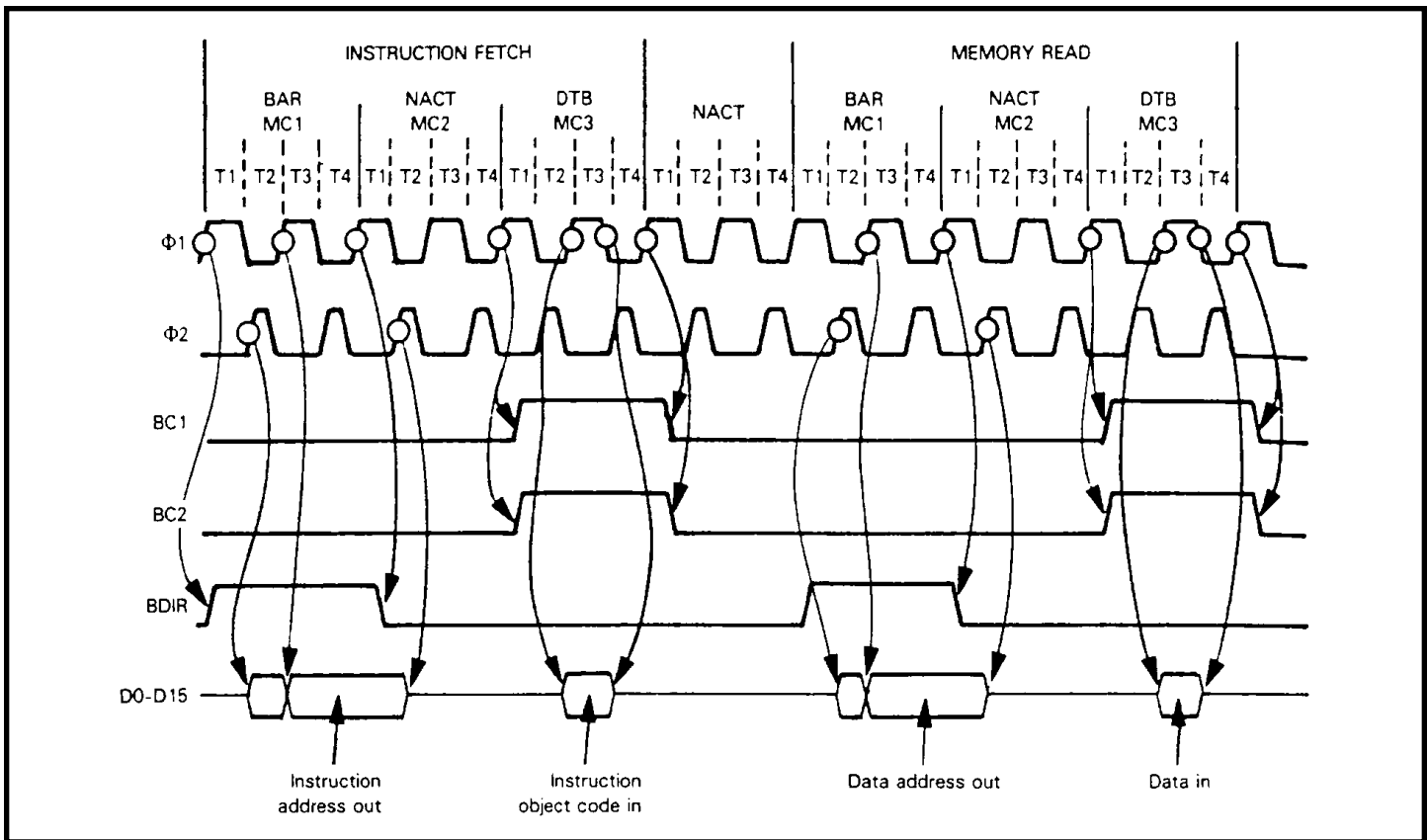


Figure 2-5. CP1600 Timing for Memory Read Instruction with Implied Memory Addressing

CP1600 INSTRUCTION TIMING AND EXECUTION

CP1600 instructions are executed as a sequence of machine cycles. Each machine cycle has four clock periods, as illustrated in Figure 2-3. Machine cycles are identified by their cycle number and by the levels of the BC1, BC2, and BDIR signals. Each of the eight level combinations is given a name, taken from Table 2-1. This name becomes the name of the machine cycle. Thus in Figure 2-4, and in subsequent instruction timing illustrations, each machine cycle is identified by a signal name from Table 2-1.

Figure 2-3 shows general case timing for data output or input on the Data/Address Bus. In between data input or output operations the bus is floated.

CP1600 MEMORY ACCESS TIMING

Figure 2-4 illustrates instruction fetch timing for a CP1600 instruction's execution. Three machine cycles are required. During the first machine cycle an address is output. Nothing happens during the second machine cycle; it is a "time spacing" machine cycle that routinely separates two CP1600 Bus access machine cycles. The object code for the accessed instruction is returned during the third machine cycle.

Figure 2-5 illustrates timing for the simplest memory read instruction's execution. In this case the data memory address is taken from one of the CPU registers. There is no difference between timing for the three machine cycles of an instruction fetch or a data memory read. As illustrated in Figure 2-5, a simple memory read instruction's execution consists of two three-machine cycle memory read operations, separated by a spacing no operation machine cycle.

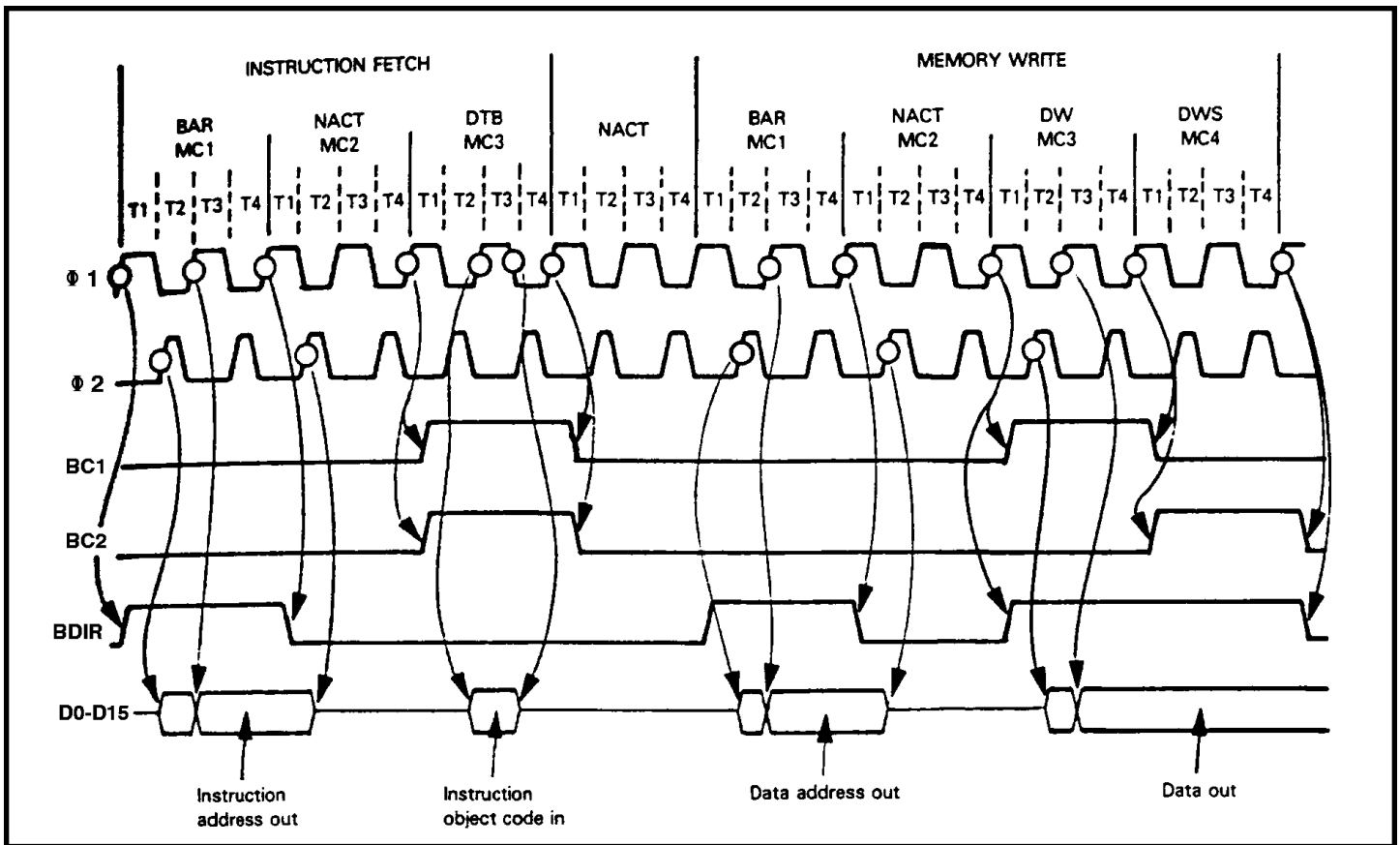
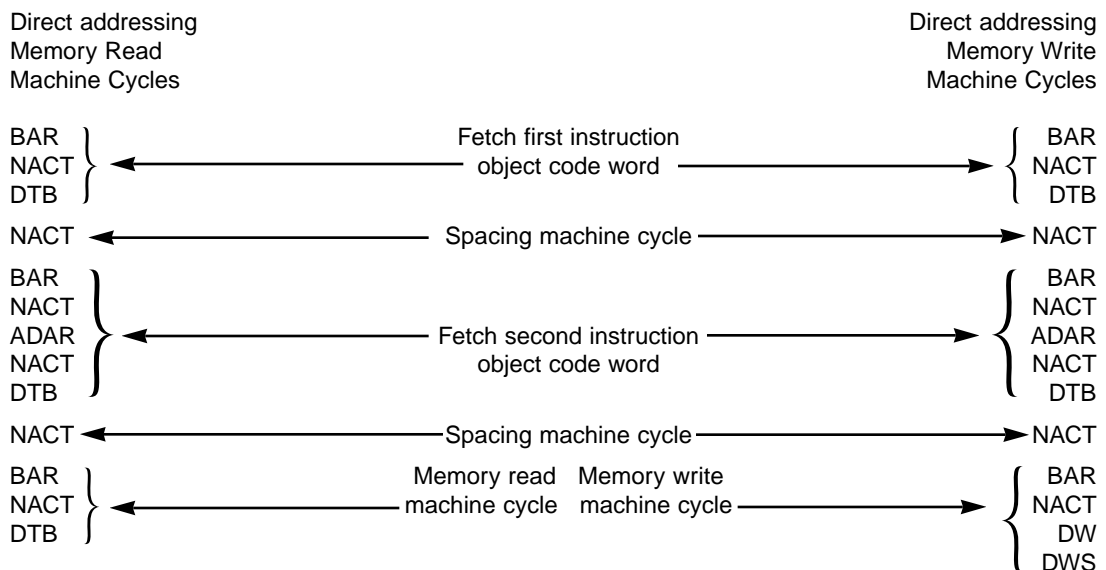


Figure 2-6 CP1600 Timing for Memory Write Instruction with Implied Memory Addressing

Figure 2-6 illustrates timing for a simple CP1600 memory write instruction execution. Data is output for two machine cycles, giving external logic ample time to respond to the data output cycle as a write strobe.

Any memory reference instruction that specifies direct memory addressing will require one three-clock-period machine cycle to fetch each word of the instruction object code; an NACT clock period will separate each machine cycle. After the first instruction fetch machine cycle an ADAR-NACT clock period combination will be inserted in the second (and third, if present) instruction fetch machine cycle. During an ADAR clock period, BC1 is high, while BC2 and BDIR are low. No other control signals are active. Thus, **for a two-word memory read or memory write instruction that specifies direct addressing, the following clock periods and machine cycles will be required for instruction execution:**



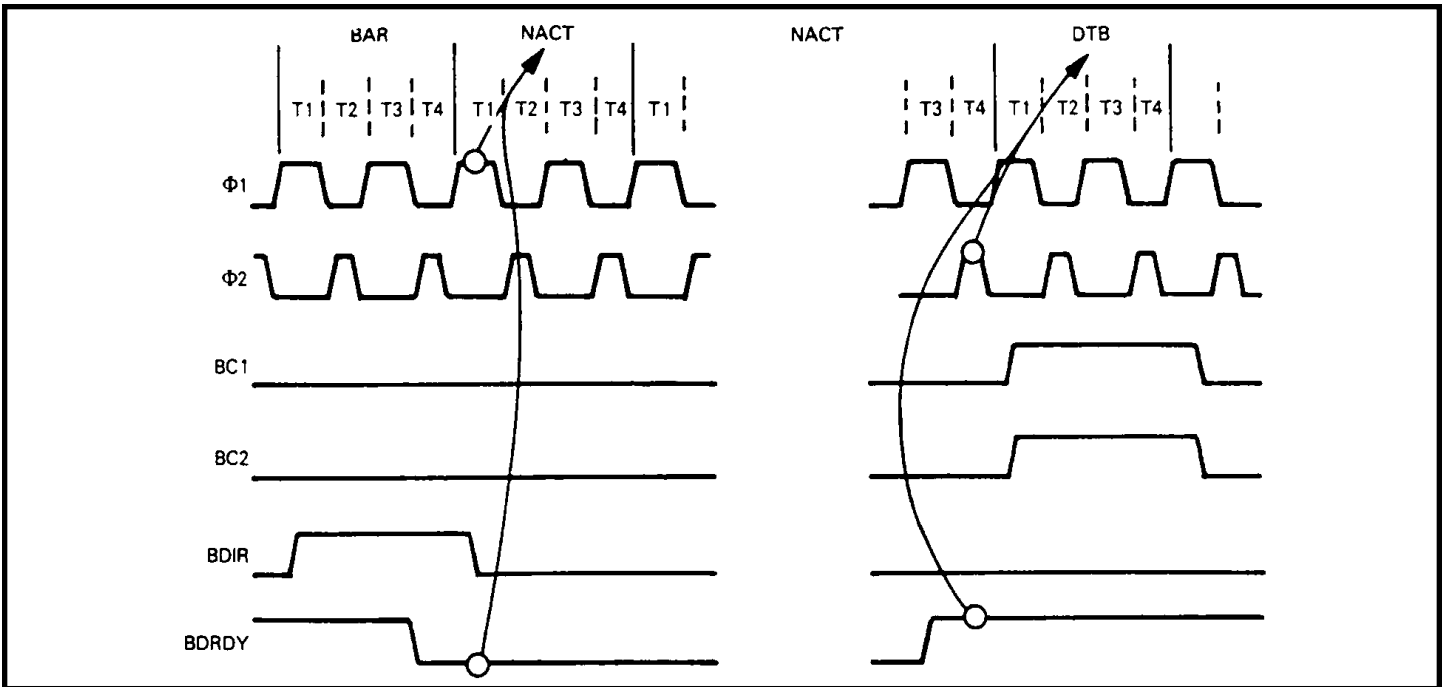


Figure 2-7 CP1600 Wait State Timing

THE CP1600 WAIT STATE

The CP1600 has a **Wait state** equivalent to those described for other microcomputers in this book. External logic that requires more time to respond to an access must input BDRDY low before the end of the BAR machine cycle, during which an address is output and the device is selected. Timing is illustrated in Figure 2-7.

If you examine Figures 2-4, 2-5, and 2-6, you will see that an address is output during a BAR machine cycle to initiate any external device access. The BAR machine cycle is always followed by an NACT machine cycle; in the middle of T1 during this NACT machine cycle, the CP1600 samples BDRDY. If BDRDY is low then a sequence of NACT machine cycles occurs in the middle of T4 for every NACT machine cycle, the CP1600 samples BDRDY again. Upon detecting BDRDY high, the CP1600 resumes instruction execution with a DTB machine cycle.

A Wait state must last for less than 40 microseconds, since the CP1600 is a dynamic device.

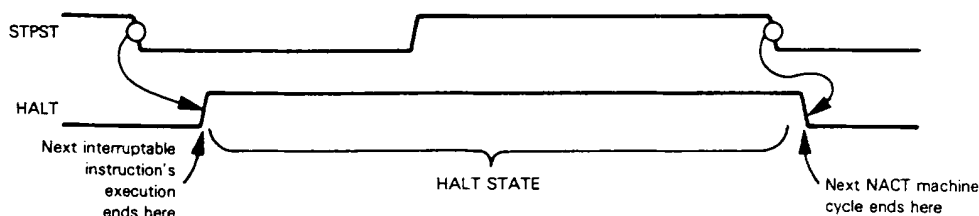
THE CP1600 HALT STATE

The CP1600 has a **Halt state** which may follow execution of the Halt instruction, or may be initiated by external logic.

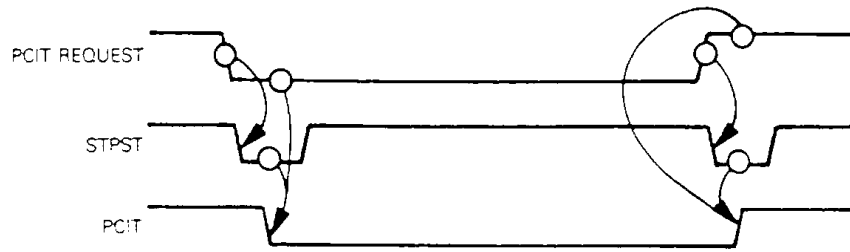
When the Halt instruction is executed, then, following the instruction fetch machine cycle, the HALT signal is output high and a sequence of NACT machine cycles is executed.

External logic initiates a Halt state by making the STPST input undergo a high-to-low transition. Following execution of the next interruptable instruction, a Halt state begins. The HALT signal is output high and a sequence of NACT machine cycles is executed.

A Halt state, whether it is initiated by execution of a Halt instruction or by a high-to-low transition of STPST, must be terminated by a high-to-low transition of STPST. This will cause the Halt state to end at the conclusion of the next NACT machine cycle. Timing for a Halt state which is initiated and terminated by STPST may be illustrated as follows:



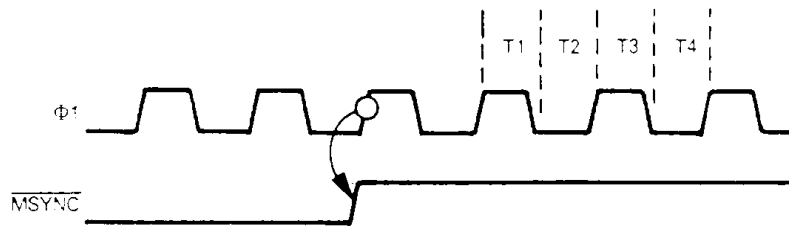
The PCIT signal as an input inhibits CP1600 Program Counter increment logic. Thus, external logic can input PCIT low — in which case the same instruction will be continuously re-executed until PCIT goes high again. However, PCIT should only change levels while the CPU has been halted. Thus, **PCIT and STPST should be used together as follows:**



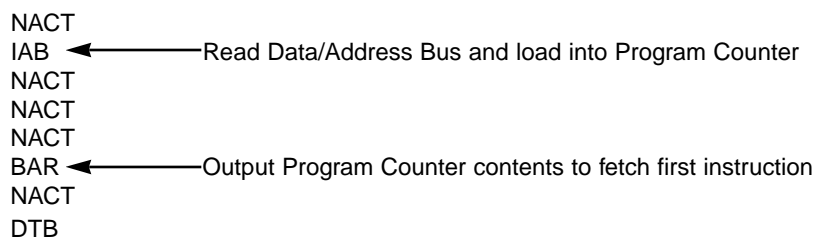
CP1600 INITIALIZATION SEQUENCE

The CP1600 is initialized by inputting the MSYNC signal low for a minimum of 10 milliseconds after power is first applied to the CPU.

MSYNC must make a low-to-high transition, marking the end of the initialization, on a rising edge of the $\Phi 1$ clock signal. On the next rising edge of $\Phi 1$, instruction execution will begin. This may be illustrated as follows:



When instruction execution begins, interrupts are disabled. The following sequence of machine cycles is executed:



During the IAB machine cycle, external logic must supply a 16-bit address at D0 - D15. Your external logic must provide this address, which in the simplest case may be 0000 by grounding the bus, or FFFF₁₆ by tying it to +5V following a startup.

The address which is input at IAB is output at BAR, initiating program execution.

CP1600 DMA LOGIC

CP1600 DMA logic is quite standard. When external logic wishes to transfer data under DMA control it inputs BUSRQ low. At the conclusion of the next interruptable instruction's execution, the CPU floats the Data/Address Bus and enters a Wait state, during which a sequence of NACT machine cycles is executed. BUSAK is output low at the beginning of the first NACT machine cycle.

The NACT machine cycles that occur during a DMA operation refresh the CPU. NACT machine cycles that occur during a Wait state do not refresh the CPU. This means that any number of NACT machine cycles can occur during a DMA break, while a Wait state must be shorter than 40 microseconds.

The DMA break ends when external logic inputs BUSRQ high again. BUSRQ is sampled during T1 of every DMA NACT machine cycle. When BUSRQ is sampled high, two additional NACT machine cycles are executed, then BUSAK is output high and normal program execution resumes.

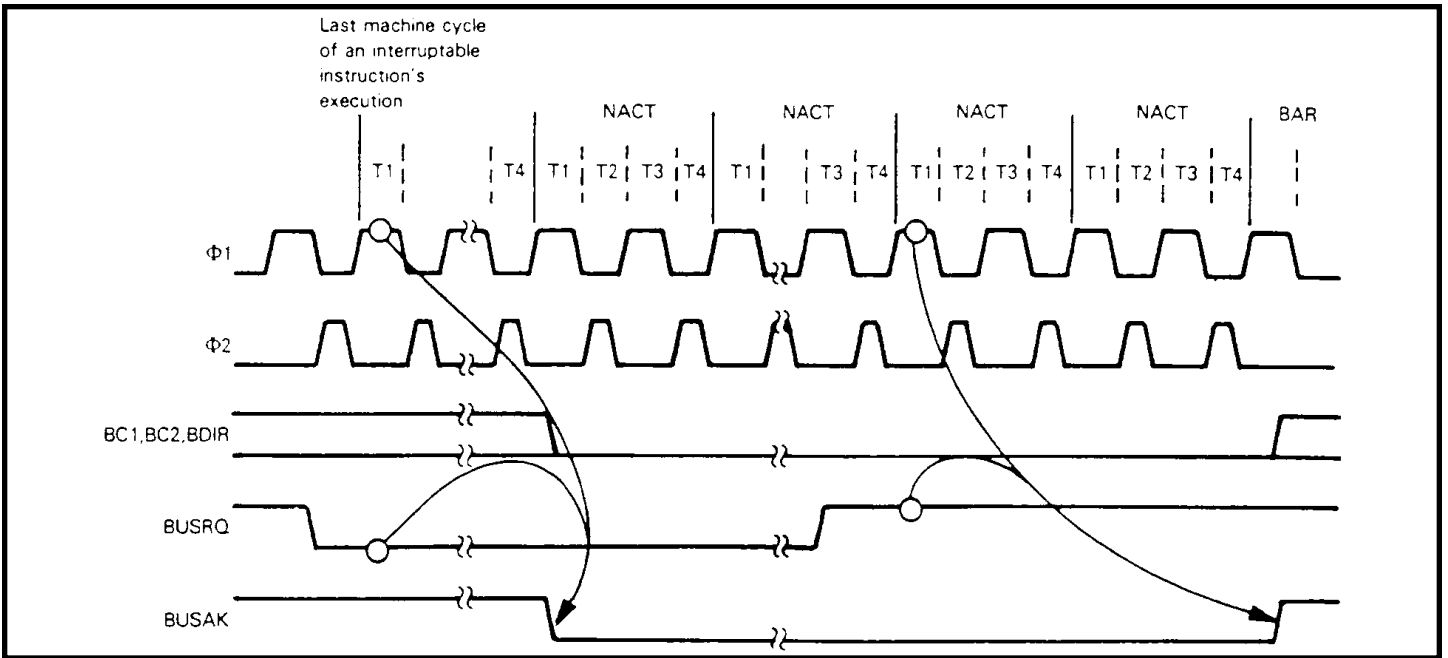


Figure 2-8. CP1600 DMA Timing

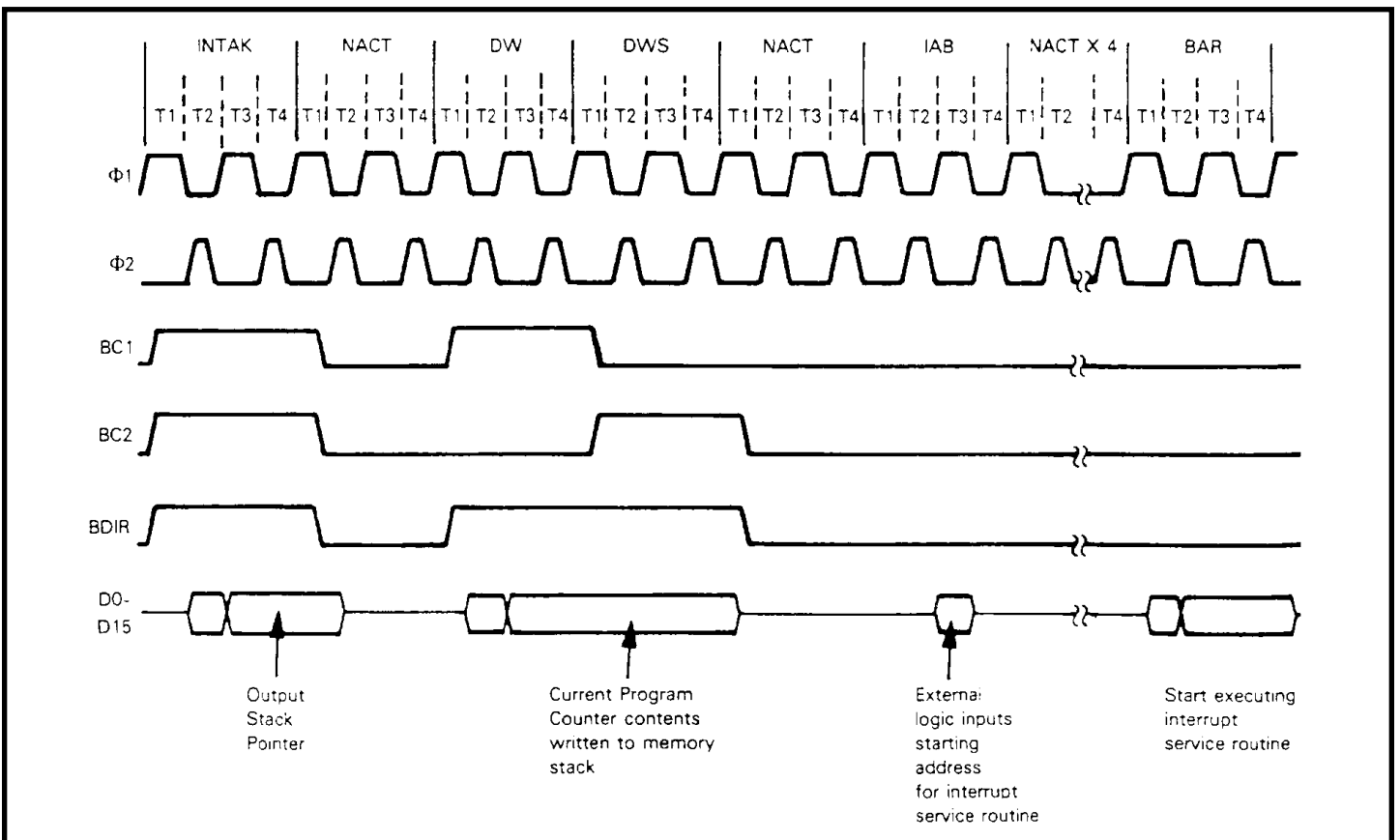


Figure 2-9. CP1600 Interrupt Service Routine Initialization

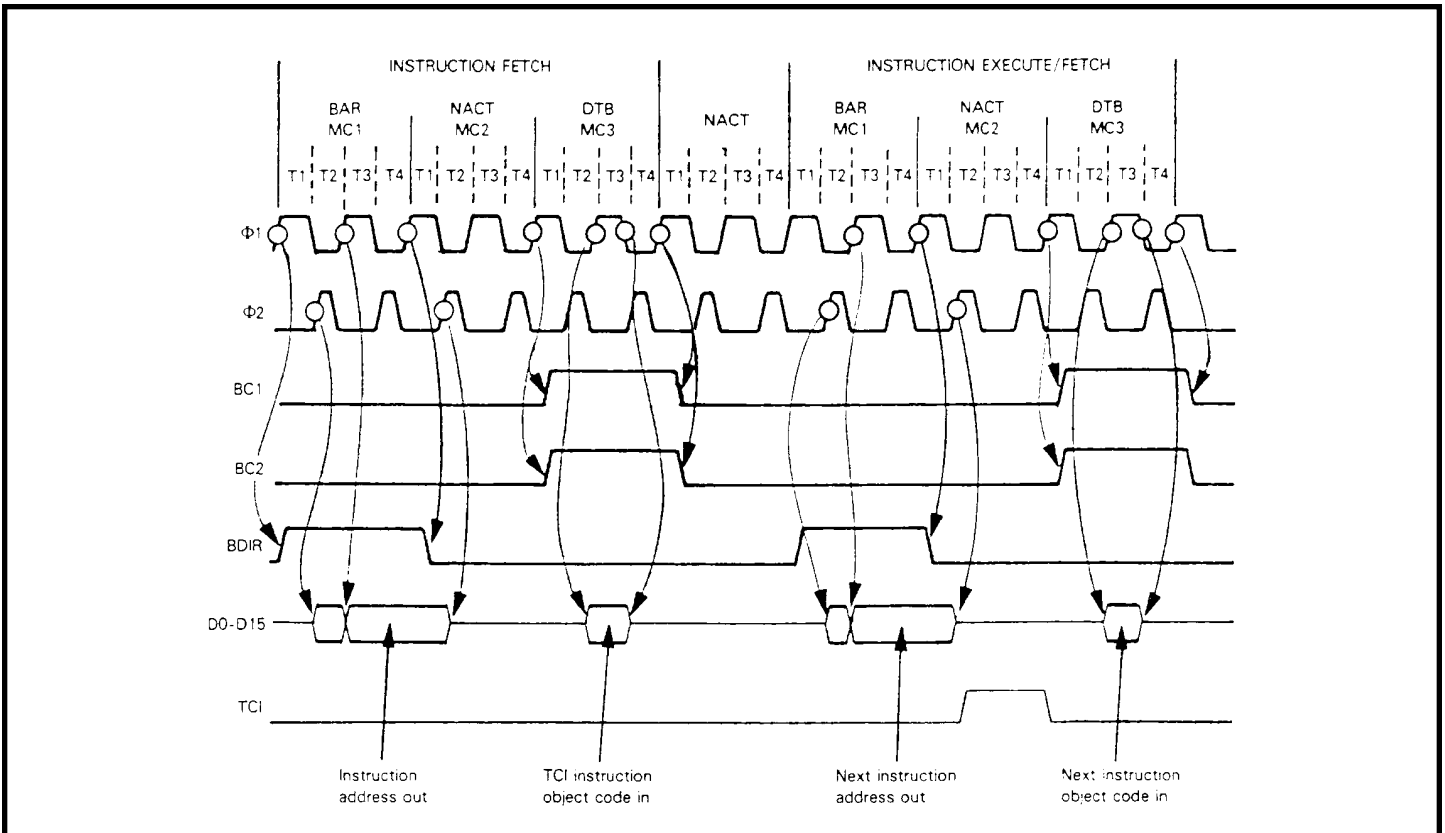


Figure 2-10 CP1600 Timing for TCI Instruction's Execution

THE CP1600 INTERRUPT LOGIC

The CP1600 uses a vectored interrupt processing system.

External logic requests an interrupt by inputting a low signal at either the INTR or INTRM pins

Following the execution of the next interruptable instruction, the CP1600 acknowledges the interrupt by pushing Register R7 contents (the Program Counter) onto the Stack; then the CP1600 outputs 111, followed by 010 at BC1, BC2, and BDIR. External logic must respond by placing 16 bits of data on the Data/Address Bus. These 16 bits of data will be loaded into Register R7, the Program Counter, thus causing program execution to branch to an interrupt service routine dedicated to the interrupt. **Timing is illustrated in Figure 2-9.**

The PCIT signal is output low following execution of a software interrupt instruction (SIN). This is the only microcomputer described in this book which allows external logic to respond to a software interrupt in this fashion. Allowing external logic to respond to a software interrupt only makes sense when you anticipate your product being used in a minicomputer-like environment. Typically, the software interrupt will interface to logic of a front panel or console. When an SIN instruction is executed, a one-machine cycle low PCIT pulse is output.

You may, if you wish end an interrupt service routine by executing a Terminate current Interrupt (TCI) instruction, in which case the TCI signal will be output high.

Timing for TCI is given in Figure 2-10.

Following an interrupt acknowledge, the interrupt service routine must execute instructions in order to disable interrupts and save the contents of registers on the Stack. The exception is Register R7, the program Counter, which is automatically pushed onto the Stack following an interrupt acknowledge.

External logic is entirely responsible for any type of interrupt priority arbitration which may occur and for the generation of the interrupt vector addressing which must be following an interrupt acknowledge.

It is quite easy to generate signals equivalent to other microcomputer system busses from the CP1600 System Bus. Therefore, you can use parts described in Volume 3 to handle CP1600 interrupt requirements.

THE CP1600 INSTRUCTION SET

The CP 1600 instruction set is relatively straight forward. Addressing modes, which we have already described, are simple, and instructions are typical of those we have seen and described for other microcomputers. Unusual features relating to addressing modes available with individual instructions are summarized in Table 2-2, which describes the CP1600 instruction set.

If you have never programmed a PDP-11 minicomputer, then you should pay particular attention to programming techniques that result from the Stack Pointer and Program Counter being accessed as general purpose registers.

A wide variety of Register Operate instructions allow you to compute data and load the result directly into Register R7, the Program Counter. In effect, these become computed Jump instructions.

The ability to manipulate Register R6, the Stack Pointer, as though it were a general purpose register means that it is easy to maintain a number of different Stacks in external read/write memory.

The Jump-to-Subroutine instruction has a minicomputer flavor to it. Rather than saving the return address on the Stack, Register R7 contents are moved to General Purpose Register R4 or R5. A number of minicomputers will save a subroutine return address in a general purpose register in this fashion. The problem with this logic is that you must execute an additional instruction within the subroutine to save the return address on the Stack if you are going to use nesting subroutines. If you are passing subroutine parameters, however, this is an excellent arrangement, for the Jump-to-Subroutine instruction places the address of the parameter list directly in a Data Counter with auto-increment. We have described the concept of parameter passing in Volume 1, Chapter 7.

Note that the CP1600 instruction set lacks a logical OR.

In Tables 2-2 and 2-4, instruction length is given in terms of "words" rather than "bytes", as we have done in previous chapters. Since only the lower 10 bits of the CP1600 object code are presently used, system configurations need not have the full 16-bit word size. Hence a "word" may be 10 to 16 bits wide, depending on the implementation.

The following notation is used in table 2-2:

ADDR	One word of direct address.
cond	Condition on which a branch may be taken. Table 1-3 lists all 14 branch conditions.
DATA	One word of immediate data.
DISP	One word displacement. See Table 2-4 for location of sign bit.
E	External branch condition.
EBCA0-3	The external branch condition address lines. EBAC0, EBAC1, EBAC2, EBAC3.
EBCI	The external branch condition input line.
LABEL	A 16-bit direct address, target of a Jump instruction. See Table 2-4 for the bit format.
PCIT	The software interrupt output line.
RB	General Purpose Register R4, R5, R6.
RD	One of the general purpose registers, used as destination for operation results.
RM	One of the general purpose registers used as a data counter, R4 or R5, if specified, is auto-incremented after memory access. R6 is incremented after a write, and decremented before a read.
RR	General Purpose Register R0, R1, R2, R3.
RS	One of the general purpose registers, used as the source of an operand.

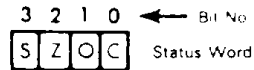
Statuses:

- S the Sign status
- C the Carry status
- Z the Zero status
- O the Overflow status

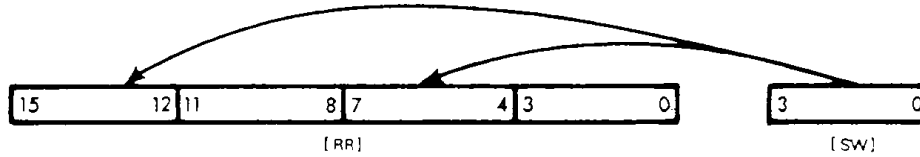
The following symbols are used in the STATUSES column

- X the status flag is affected by the operation
a blank means the status flag is not affected
- 0 the operation clears the status flag
- 1 the operation sets the status flag
- 2 the overflow flag is affected only on 2-bit shifts or rotates

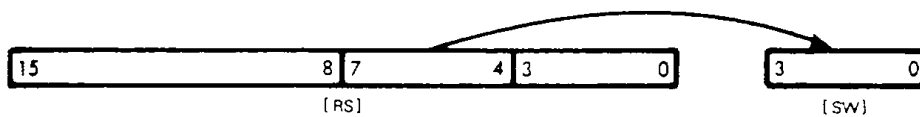
SW The Status Word, whose bits correspond to the condition of the status flags in the following way:



When the status word is copied into a register, it goes to the upper half of each byte:



When the status word is loaded from a register, it comes from the upper half of the lower byte:



$x\langle y,z\rangle$ Bits y through z of the Register x For example, $R7\langle 15,8\rangle$ represents the upper byte of Program Counter

$(,2)$ Indicates that the operand “2” is optional

$\text{—} \text{—} \text{—}$ A low pulse

$[]$ Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated registers contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.

$[[]]$ Implied memory addressing, the contents of the memory location designated by the contents of a register.

\wedge Logical AND

∇ Logical Exclusive-OR

\vee Logical OR

\pm Addition or subtraction of a displacement. depending on the sign bit in the object code.

\leftarrow Data is transferred in the direction of the arrow

Table 2-2. CP1600 Instruct Set Summary

TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
PRIMARY I/O AND MEMORY REFERENCE	MVI	ADDR, RD	2					$[RD] \leftarrow [ADDR]$ Loads register from memory, using direct addressing.
	MVI@	RM, RD	1					$[RD] \leftarrow [[RM]]$ Loads register from memory, using implied addressing.
	MVO	RS, ADDR	2					$[ADDR] \leftarrow [RS]$ Store register to memory, using direct addressing.
	MVO@	RS, RM	1					$[[RM]] \leftarrow [RS]$ Store register to memory, using implied addressing. If RS = R4, R6, or R7, then RS = RM is not supported.
SECONDARY I/O AND MEMORY REFERENCE	ADD	ADDR, RD	2	X	X	X	X	$[RD] \leftarrow [RD] + [ADDR]$ Add memory contents to register, using direct addressing.
	ADD@	RM, RD	1	X	X	X	X	$[RD] \leftarrow [RD] + [[RM]]$ Add memory contents to register, using implied addressing.
	SUB	ADDR, RD	2	X	X	X	X	$[RD] \leftarrow [RD] - [ADDR]$ Subtract memory contents from register, using direct addressing.
	SUB@	RM, RD	1	X	X	X	X	$[RD] \leftarrow [RD] - [[RM]]$ Subtract memory contents from register, using implied addressing.
	CMP	ADDR, RS	2	X	X	X	X	$[RS] - [ADDR]$ Compare memory contents with registers, using direct addressing. Only the status flags are affected.
	CMP@	RM, RS	1	X	X	X	X	$[RS] - [[RM]]$ Compare memory contents with registers, using implied addressing. Only the status flags are affected.
	AND	ADDR, RD	2	X	X			$[RD] \leftarrow [RD] \wedge [ADDR]$ AND memory contents with those of register, using direct addressing.
	AND@	RM, RD	1	X	X			$[RD] \leftarrow [RD] \wedge [[RM]]$ AND memory contents with those of register, using implied addressing.
	XOR	ADDR, RD	2	X	X			$[RD] \leftarrow [RD] \vee [ADDR]$ Exclusive-OR memory contents with those of register, using direct addressing.
	XOR@	RM, RD	1	X	X			$[RD] \leftarrow [RD] \vee [[RM]]$ Exclusive-OR memory contents with those of register, using implied addressing.

Table 2-2. CP1600 Instruct Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
IMMEDIATE	MVII	DATA, RD	2					$[RD] \leftarrow \text{DATA}$ Load immediate to specified register.
	MVOI	RS, DATA	2					$[[R7]+1] \leftarrow \text{RS}$ Store contents of specified register in immediate field of MVOI instruction. This is only possible if program is read/write memory (rather than ROM).
IMMEDIATE OPERATE	ADDI	DATA, RD	2	X	X	X	X	$[RD] \leftarrow [RD] + \text{DATA}$ Add immediate to specified register.
	SUBI	DATA, RD	2	X	X	X	X	$[RD] \leftarrow [RD] - \text{DATA}$ Subtract immediate data from specified register.
	CMPI	DATA, RS	2	X	X	X	X	$[RD] - \text{DATA}$ Compare immediate data with contents of specified register. Only the status flags are affected.
	ANDI	DATA, RD	2	X	X			$[RD] \leftarrow [RD] \wedge \text{DATA}$ AND immediate data with contents of specified register.
	XORI	DATA, RD	2	X	X			$[RD] \leftarrow [RD] \vee \text{DATA}$ Exclusive-OR immediate data with contents of specified register.
JUMP	J	LABEL	3					$[R7] \leftarrow \text{LABEL}$ Jump to given address.
	JR	RS	1	X	X			$[R7] \leftarrow [RS]$ Jump to address contained in specified register.
	JSR	RB, LABEL	3					$[RB] \leftarrow [R7], [R7] \leftarrow \text{LABEL}$ Jump to given address, saving Program Counter in R4, R5, or R6.
	B	DISP	2					$[R7] \leftarrow [R7] + 2 \pm \text{DISP}$ Branch relative to Program Counter contents.
BRANCH ON CONDITION	Bcond	DISP	2					If cond is true, $[R7] \leftarrow [R7] + 2 \pm \text{DISP}$ Branch relative on given condition; otherwise, execute next sequential instruction.
	BEXT	DISP, E	2					$\text{EBCA0-3} \leftarrow \text{E};$ If $\text{EBCI} = 1$, $[R7] \leftarrow [R7] + 2 \pm \text{DISP}$ Branch relative if external condition is true.

Table 2-2. CP1600 Instruct Set Summary (Continued)

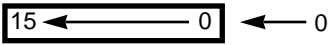
TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
REGISTER-REGISTER MOVE AND OPERATE	MOVR	RS, RD	1	X	X			$[RD] \leftarrow [RS]$ Move contents of source register to destination register.
	ADDR	RS, RD	1	X	X	X	X	$[RD] \leftarrow [RS] + [RD]$ Add contents of specified registers.
	SUBR	RS, RD	1	X	X	X	X	$[RD] \leftarrow [RD] - [RS]$ Subtract registers of source register from thoes of destination register.
	CMPR	RS, RD	1	X	X	X	X	$[RD] - [RS]$ Compare registers' contents. Only the status flags are affected.
	ANDR	RS, RD	1	X	X			$[RD] \leftarrow [RD] \wedge [RS]$ AND contents of specified registers
	XORR	RS, RD	1	X	X			$[RD] \leftarrow [RD] \vee [RS]$ Exclusive-OR contents of specified registers.
REGISTER OPERATE	CLRR	RD	1	0	1			$[RD] \leftarrow [RD] \vee [RD]$ Clear specified register.
	TSTR	RS	1	X	X			$[RD] \leftarrow [RS]$ Test contents of specified register.
	INCR	RD	1	X	X			$[RD] \leftarrow [RD] + 1$ Increments contents of specified register.
	DECR	RD	1	X	X			$[RD] \leftarrow [RD] - 1$ Decrements contents of specified register.
	COMR	RD	1	X	X			$[RD] \leftarrow \overline{[RD]}$ Complements contents of specified register (ones complement).
	NEGR	RD	1	X	X	X	X	$[RD] \leftarrow 00_{16} - [RD]$ Negates contents of specified register (twos complement).
	ADCR	RD	1	X	X	X	X	$[RD] \leftarrow [RD] + [C]$ Add Carry bit to specified register contents.
	SLL	RR(,2)	1	X	X			 $[RR]$ Shift logical left one or two bits, clearing bit 0 (and bit 1 is shifting twice).

Table 2-2. CP1600 Instruct Set Summary (Continued)

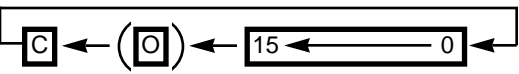
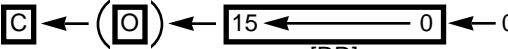


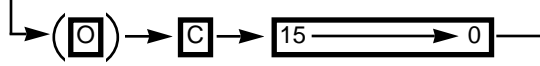
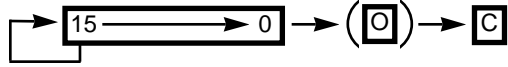
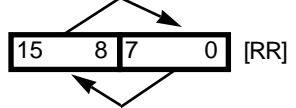
TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
REGISTER OPERATE (CONTINUED)	RLC	RR(,2)	1	X	X	X	2	 <p>[RR]</p> <p>Rotate left one bit through Carry, or rotate two bits left through Overflow and Carry.</p>
	SLLC	RR(,2)	1	X	X	X	2	 <p>[RR]</p> <p>Shift logical left one bit into Carry, clearing bit 0, or shift left two bits into Overflow and Carry, clearing bits 0 and 1.</p>
	SLR	RR(,2)	1	X	X			 <p>[RR]</p> <p>Shift logical right one or two bits, clearing bit 15 (and bit 14 if shifting twice).</p>
	SAR	RR(,2)	1	X	X			 <p>[RR]</p> <p>Shift arithmetic right one or two bits, copying high order bit.</p>
	RRC	RR(,2)	1	X	X	X	2	 <p>[RR]</p> <p>Rotate right one bit through Carry, or rotate two bits right through Overflow and Carry.</p>
	SARC	RR(,2)	1	X	X	X	2	 <p>[RR]</p> <p>Shift arithmetic right one bit into Carry, or two bits into Overflow and Carry.</p>
	SWAP	RR(,2)	1	X	X			 <p>[RR]</p> <p>Swap bytes of register once, or twice.</p>

Table 2-2. CP1600 Instruct Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	WORDS	STATUSES				OPERATION PERFORMED
				S	Z	C	O	
STACK	PSHR	RS	1					Separate mnemonics for MVO@ RS,R6
	PULR	RD	1					Seperate mnemonics for MVI@ R6,RD
INTERRUPT	SIN	(2)	1					$PCIT \leftarrow \square$ Software interrupt
	EIS		1					Enable interrupt system.
	DIS		1					Disable interupt system.
	TCI		1					Terminate current interrupt.
	JE	LABEL	3					Jump to given address and enble interrupt system.
	JD	LABEL	3					Jump to give address and disable interrupt system.
	JSRE	RB, LABEL	3					Jump to given address, saving Program Counter in R4, R5, or R6, and enable interrupt system.
JSRD	RB, LABEL	3					Jump to given address, saving Program Counter in R4, R5, or R6, and disable interrupt system.	
STATUS	GSWD	RD	1					$[RD <15,12>] \leftarrow [SW]; [RD <7,4>] \leftarrow [SW]$ Place Status Word in upper half of each byte of the specified register. RD may be R0, R1, R2, or R3.
	RSWD	RS	1	X	X	X	X	$[SW] \leftarrow [RS <7,4>]$ Load Status Word from bits 7 through 4 of the specified register.
	CLRC		1			0		$[C] \leftarrow 0$ Clear Carry.
	SETC		1			1		$[C] \leftarrow 1$ Set Carry.
	NOPP		2					No Operation
	NOP	(2)	1					Halt after executing next instruction.
	HLT		1					Set double byt data mode for next instruction, which mush be one of the following types: Primary or secondary I/O or memory reference Immediate or immediate operate
	SDBD		1					If implied addressing throught R1, R2, or R3 is used, the same byte will be accessed twice; addressing through R4, R5, or R7 will give bytes from the addressed location and that addressed after auto-increment. Direct addressing and Stack addressing are not allowed in double byte mode.

Table 2-3. CP1600 Branch Conditions and Corresponding Codes

MNEMONIC	BRANCH CONDITION	OBJECT CODE DESIGNATION
C LGT	C = 1 Carry (Logical greater than)	0001
NC LLT	C = 0 No Carry (Logical less than)	1001
OV	O = 1 Overflow	0010
NOV	O = 0 No Overflow	1010
PL	S = 0 Plus	0011
MI	S = 1 Minus	1011
ZE	Z = 1 Zero (equal)	0100
NZE	Z = 0	1100
NEQ	Nonzero (not equal)	
LT	$S \neq O = 1$ Less than	0101
GE	$S \neq O = 0$ Greater than or equal	1101
LE	$Z \vee (S \neq O) = 1$ Less than or equal	0110
GT	$Z \vee (S \neq O) = 0$ Greater than	1110
USC	$C \neq S = 1$ Unequal sign and carry	0111
ESC	$C \neq S = 0$ Equal sign and carry	1111

The following notation is used in Table 2-4:

Where ten digits are shown, they are the ten low-order bits of a 10 to 16-bit word (Word size depends on the system implementation.) Where four digits are shown, they represent the hexadecimal notation for an entire word (10 to 16 bits).

bb Two bits indicating one of the first three general purpose registers:

- 00 = R0
- 01 = R1
- 10 = R2

cccc Four bits giving branch condition, as shown in Table 2-3.

ddd Three bits indicating a destination register, RD:

- 000 = R0
- 001 = R1
- 010 = R2
- 011 = R3
- 100 = R4
- 101 = R5
- 110 = R6
- 111 = R7

eeee Four bits giving the external branch condition, E. Control signals EBCA0-EBCA3 reflect the state of these four bits.

llll One word of immediate data (10 or 16 bits)

mmmm	Three bits indicating a Data Counter Registered RM: 000 = R0 001 = R1 010 = R2 011 = R3 100 = R4 101 = R5 110 = R6 111 = R7
m	One bit indicating the number of rotates or shifts 0 one bit positions 1 two bit positions
p	One bit of immediate address
P	One hexadecimal digit (4 bits) of immediate address
rr	Two bits indicating one of the first four general purpose registers. 00 = R0 01 = R1 10 = R2 11 = R3
sss	Three bits indication a source register, RD: 000 = R0 001 = R1 010 = R2 011 = R3 100 = R4 101 = R5 110 = R6 111 = R7
z	Sign of the displacement 0 add the displacement to PC contents 1 subtract the displacement from PC contents

In the "Machine Cycles" column, when two numbers are given with one slash between them (e.g., 7/9), execution time depends on whether or not a branch is taken. When two numbers are given, separated by two slashes (such as 8//11), execution time depends on which register contains the implied address.

THE BENCH MARK PROGRAM

For the CP1600 our benchmark program may be illustrated as follows:

	MVII	IOBUF,R4	LOAD THE I/O BUFFER STARTING ADDRESS INTO R4
	MVII	TABLE,R1	LOAD THE TABLE STARTING ADDRESS INTO R1
	MVI@	R1,R5	LOAD ADDRESS OF FIRST FREE TABLE WORD INTO R5
	MVII	CNT,R2	LOAD WORD COUNT INTO R2
LOOP	MVI@	R4,R0	LOAD NEXT DATA WORD FROM IOBUF
	MVO@	R0,R5	STORE IN NEXT TABLE WORD
	DECR	R2	DECREMENT WORD COUNT
	BNZE	LOOP	RETURN IF NOT END
	MVO@	R5,R1	RETURN ADDRESS OF NEXT FREE TABLE BYTE

This benchmark program makes very few assumptions. The input table IOBUF and the data table TABLE can have any length, and can reside anywhere in memory. The address of the first word in TABLE is stored in the first word of the TABLE.

Table 2-4. CP1600 Instruction Set Object Codes

INSTRUCTION	OBJECT CODE	WORDS	MACHINE CYCLES	INSTRUCTION	OBJECT CODE	WORDS	MACHINE CYCLES
ADCR RD	0000101ddd	1	6	JSRE RB,LABEL	0000 bbpppppp01 PPPP	3	12
ADD ADDR,RD	1011000ddd PPPP	2	10	MOVR RS,RD	0010sssddd	1	5//7
ADD@ RM,RD	1011mmmddd	1	8//11	MVI ADDR,RD	1010000ddd PPPP	2	10
ADDI DATA,RD	1011111ddd IIII	2	8	MVI@ RM,RD	1010mmmddd	1	8//11
ADDR RS,RD	0011sssddd	1	6	MVII DATA,RD	101011ddd IIII	2	5
AND ADDR,RD	1110000ddd PPPP	2	10	MVO RS,ADDR	1001000sss PPPP	2	11
AND@ RM,RD	1110mmmddd	1	8//11	MVO@ RS,RM	1001mmmsss	1	9
ANDI DATA,RD	1110111ddd IIII	2	8	MVOI RS,DATA	1001111ssss IIII	2	9
ANDR RS,RD	0110ssddd	1	6	NEGR RD	0000100ddd	1	6
B DISP	1000z00000 PPPP	2	7/9	NOP (2)	000011010m	1	5
Bcond DISP	1000z0cccc PPPP	2	7/9	NOPP	1000z01000 PPPP	2	7
BEXT DISPE	1000z1eeee PPPP	2	7/9	PSHR RS	1001110sss	1	9
CLRC	0006	1	4	PULR RD	1010110ddd	1	11
CLRR RD	0111dddddd	1	6	RLC RR,(2)	0001010mrr	1	6/8
CMP ADDR,RS	1101000sss PPPP	2	10	RRC RR,(2)	0001110mrr	1	6/8
CMP@ RM,RS	1101mmmsss	1	8//11	RSWD RS	0000111sss	1	6
CMPI DATA,RS	110111sss IIII	2	8	SAR RR,(2)	0001101mrr	1	6/8
CMPR RS,RD	0101sssddd	1	6	SARC RR,(2)	0001111mrr	1	6/8
COMR RD	0000011ddd	1	6	SDBD	0001	1	4
DECR	0000010ddd	1	6	SETC	0007	1	4
DIS	0003	1	4	SIN (2)	000011011m	1	6
EIS	0002	1	4	SLL RR,(2)	0001001mrr	1	6/8
GSWD RR	00001100rr	1	6	SLLC RR,(2)	0001011mrr	1	6/8
HLT	0000	1	4	SLR (RR,2)	0001100mrr	1	6/8
INCR	0000001ddd	1	5	SUB ADDR,RD	1100000ddd PPPP	2	10
J LABEL	0004 11pppppp00 PPPP	3	12	SUB@ RM,RD	1100mmmddd	1	8//11
JD LABEL	0004 11pppppp10 PPPP	3	12	SUBT DATA,RD	1100111ddd IIII	2	8
JE LABEL	0004 11pppppp01 PPPP	3	12	SUBR RS,RD	0100sssddd	1	6
JR RS	0010sss111	1	7	SWAP RR,(2)	0001000mrr	1	6/8
JSR RB,LABEL	0004 bbpppppp00 PPPP	3	12	TCI	0005	1	4
JSRD RB,LABEL	0004 bbpppppp10 PPPP	3	12	TSTR RS	0010ssssss	1	6//7
				XOR ADDR,RD	1111000ddd PPPP	2	10
				XOR@ RM,RD	111mmmddd	1	8//11
				XORI DATA,RD	1111111ddd IIII	2	8
				XORR RS,RD	0111sssddd	1	6

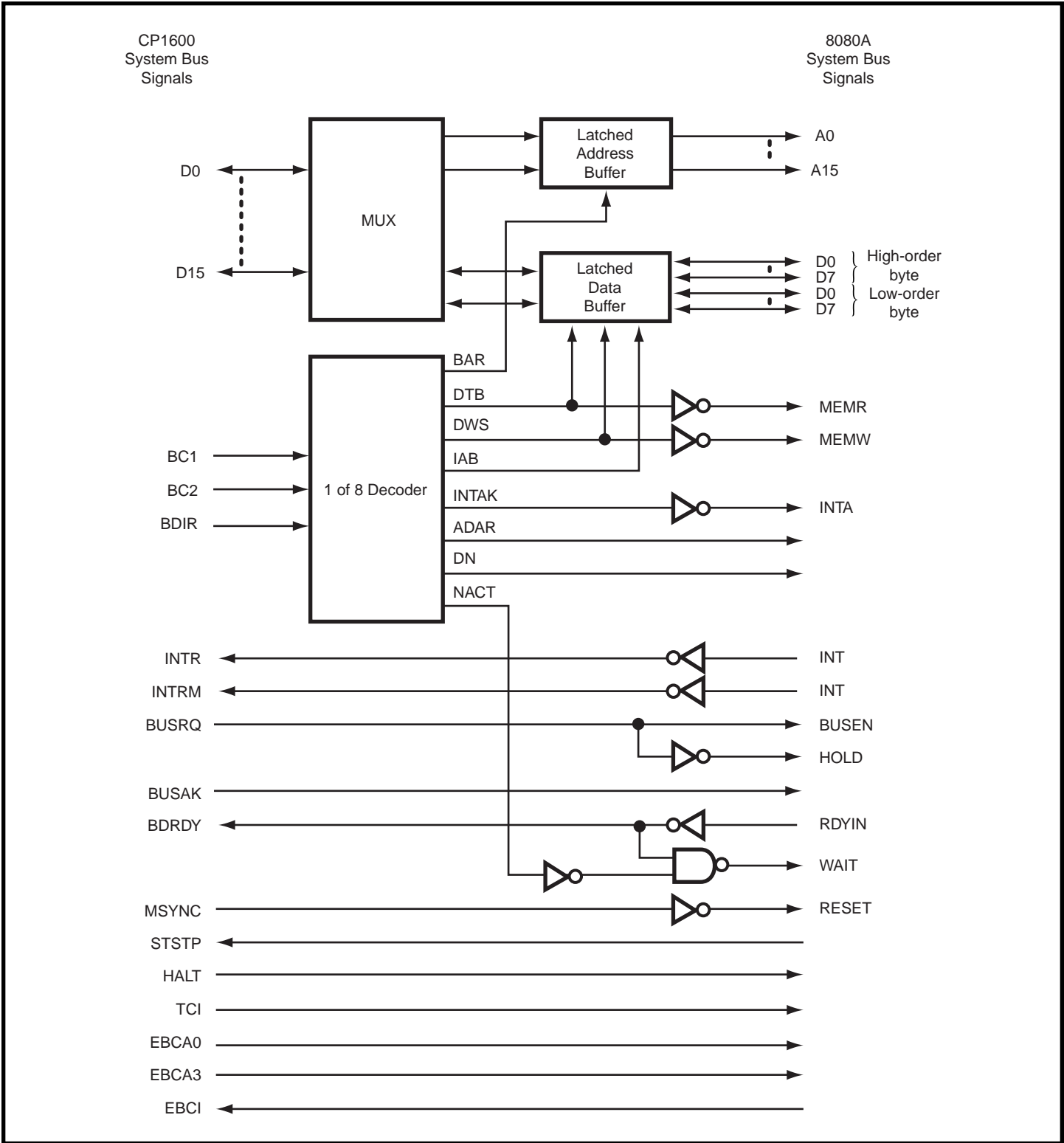


Figure 2-11, CP1600 to 8080A Bus Conversion

SUPPORT DEVICES THAT MAY BE USED WITH THE CP1600

A CP1600 microcomputer system with any significant capabilities will use support of some other microprocessor. Parallel I/O capability is available with the CP1680, (described next), but priority interrupt logic, DMA logic, and serial I/O logic, to mention just a few common options, may need additional support devices. Fortunately, **it is quite easy to generate an 8080A-compatible system bus from the CP1600 system bus. Logic is illustrated in Figure 2-11.**

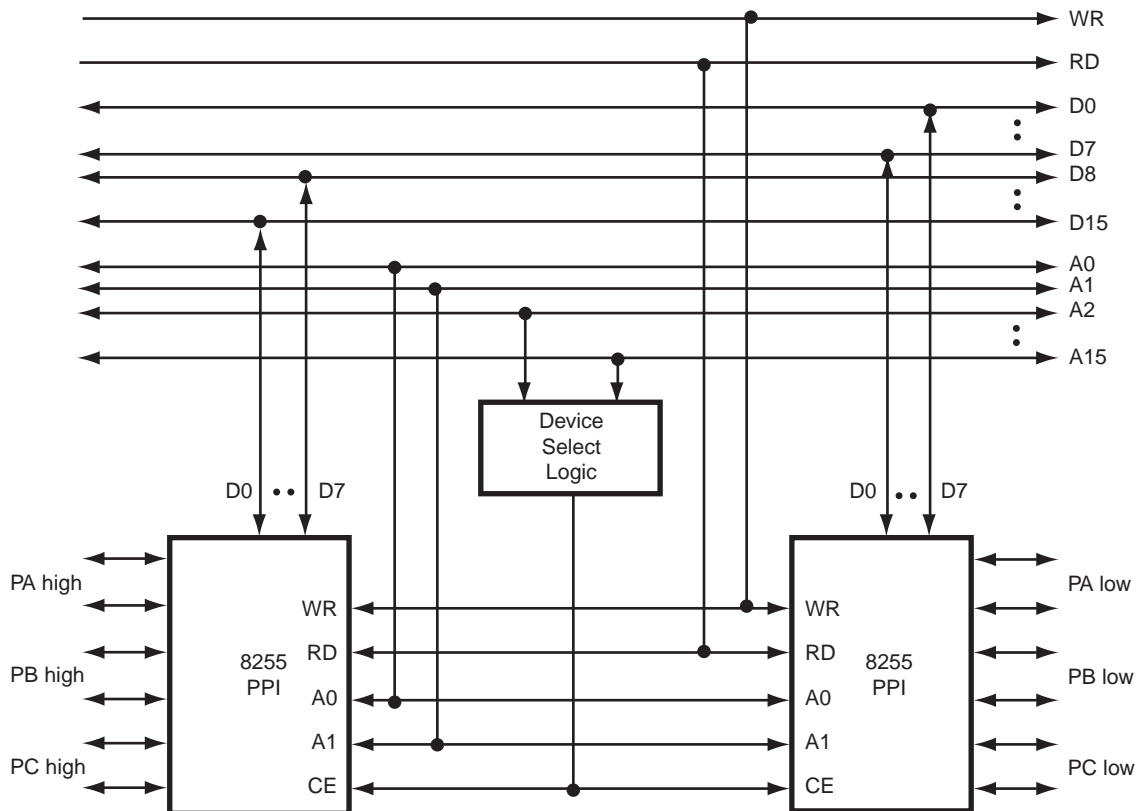
The CP1600A is the fastest version of the CP1600 CPU: it runs with a 500 nanosecond machine cycle. The CP1600 machine cycle is equivalent to an 8080A clock period. Since the standard 8080A clock period is also 500 nanoseconds, no speed conflicts will arise.

The bus-to-bus interface logic illustrated in Figure 2-1 is self-evident, with the exception of bus demultiplexing logic. The CP1600 Data/Address bus is shown buffered by a demultiplexing buffer that is connected to two latched buffers. One of the latched buffers accepts the demultiplexer outputs only when a valid address is being output, as identified by BAR high. The second latched buffer may be a bidirectional latched buffer, or it may be two unidirectional latched buffers. Three latching strobes are required: DTB, IAB, and DWS.

DTB and IAB are data input strobes. DTB strobes data input that is to be interpreted as data, while IAB strobes data input that is to be interpreted as an address. So far as external logic is concerned, both of these signals are simple data input strobes. We could therefore generate a single data input strobe as the OR of DTB and IAB. When this data input strobe is high, information on the 8080A System Bus side of the latched data buffer must be input to the buffer; this data must simultaneously be transmitted to the multiplexer.

DWS is the data output strobe. When high, this signal must strobe data from the multiplexer to the latched data buffer; this latched data must immediately appear at the 8080A System Bus side of the latched data buffer.

Since the CP1600 uses a 16-bit Data Bus, you will probably have to generate two external device data busses: a high-order byte bus and a low-order byte bus. All external devices that transmit or receive parallel data must be present in duplicate. For example, were 8255 parallel interface devices to be present, the following connections would be required:



Osbourne/Mcgraw Hill is the copyright holder of this document.
Please email any corrections to tlindner@ix.netcom.com