4.0 CPU TIMING

The Z80-CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

Memory read or write

I/O device read or write

Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T states and the basic operations are referred to as M (for machine) cycles. Figure 4.0-0 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five or six T states long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles. In section 7, the exact timing for each instruction is specified.

BASIC CPU TIMING EXAMPLE

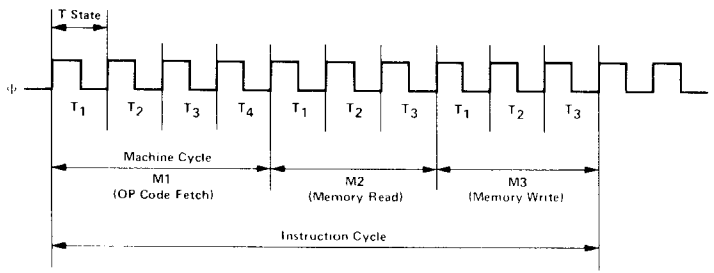


FIGURE 4.0-0

All CPU timing can be broken down into a few very simple timing diagrams as shown in Figure 4.0-1 through 4.0-7. These diagrams show the following basic operations with and without wait states (wait states are added to synchronize the CPU to slow memory or I/O devices).

4.0-1. Instruction OP code fetch (M1 cycle)
4.0-2. Memory data read or write cycles
4.0-3. I/O read or write cycles
4.0-4. Bus Request/Acknowledge Cycle
4.0-5. Interrupt Request/Acknowledge Cycle
4.0-6. Non maskable Interrupt Request/Acknowledge Cycle
4.0-7. Exit from a HALT instruction

INSTRUCTION FETCH

Figure 4.0-1 shows the timing during an M1 cycle (OP code fetch). Notice that the PC is placed on the address bus at the beginning of the M1 cycle. One half clock time later the $\overline{MREQ}$ signal goes active. At this time the address to the memory has had time to stabilize so that the falling edge of $\overline{MREQ}$ can be used directly as a chip enable clock to dynamic memories. The $\overline{RD}$ line also goes active to indicate that the memory read data should be enabled onto the CPU data bus. The CPU samples the data from the memory on the data bus with the rising edge of the clock of state T3 and this same edge is used by the CPU to turn off the $\overline{RD}$ and $\overline{MREQ}$ signals. Thus the data has already been sampled by the CPU before the $\overline{RD}$ signal becomes inactive. Clock state T3 and T4 of a fetch cycle are used to refresh dynamic memories. (The CPU uses this time to decode and execute the fetched instruction so that no other operation could be performed at this time). During T3 and T4 the lower 7 bits of the address bus contain a memory refresh address and the $\overline{RFSH}$ signal becomes active to indicate that a refresh read of all dynamic memories should be accomplished. Notice that a $\overline{RD}$ signal is not generated during refresh time to prevent data from different memory segments from being gated onto the data bus. The $\overline{MREQ}$ signal during refresh time should be used to perform a refresh read of all memory elements. The refresh signal can not be used by itself since the refresh address is only guaranteed to be stable during $\overline{MREQ}$ time.

INSTRUCTION OP CODE FETCH

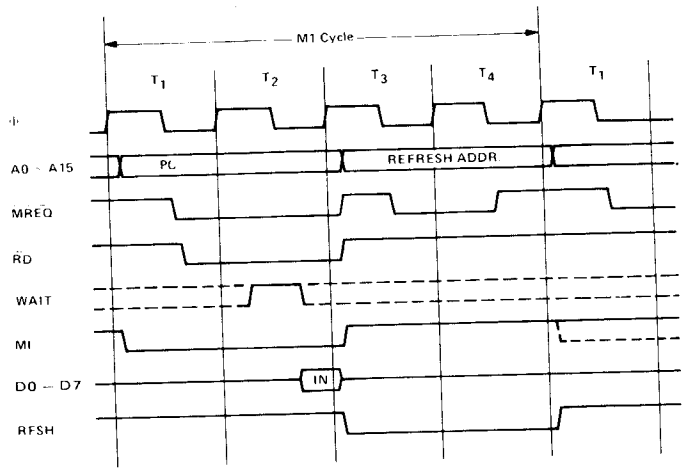


FIGURE 4.0-1

Figure 4.0-1A illustrates how the fetch cycle is delayed if the memory activates the $\overline{WAIT}$ line. During T2 and every subsequent Tw, the CPU samples the $\overline{WAIT}$ line with the falling edge of Φ. If the $\overline{WAIT}$ line is active at this time, another wait state will be entered during the following cycle. Using this technique the read cycle can be lengthened to match the access time of any type of memory device.

INSTRUCTION OP CODE FE

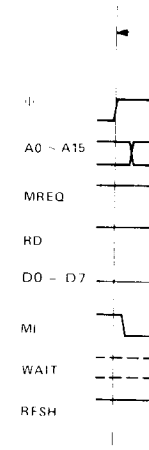


FIGURE 4.0-1A

MEMORY READ OR

Figure 4.0-2 illustrate
fetch (M1 cycle). Thes
requested by the mem
the same as in the fetc
active when the addre
dynamic memories. Th
be used directly as a
more the $\overline{WR}$ signal go
are changed so that
memory type will be m

MEMORY READ OR WRITE C

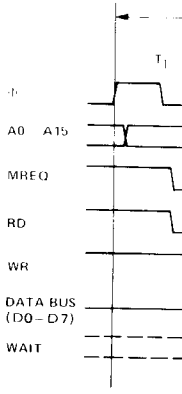


FIGURE 4.0-2

Notice that the PC is
clock time later the
had time to stabilize
ple clock to dynamic
read data should be
memory on the data
e is used by the CPU
n sampled by the CPU
etch cycle are used to
execute the fetched
he). During T3 and T4
s and the $\overline{RFSH}$ signal
ries should be accom-
to prevent data from
e $\overline{MREQ}$ signal during
elements. The refresh
uaranteed to be stable

INSTRUCTION OP CODE FETCH WITH WAIT STATES

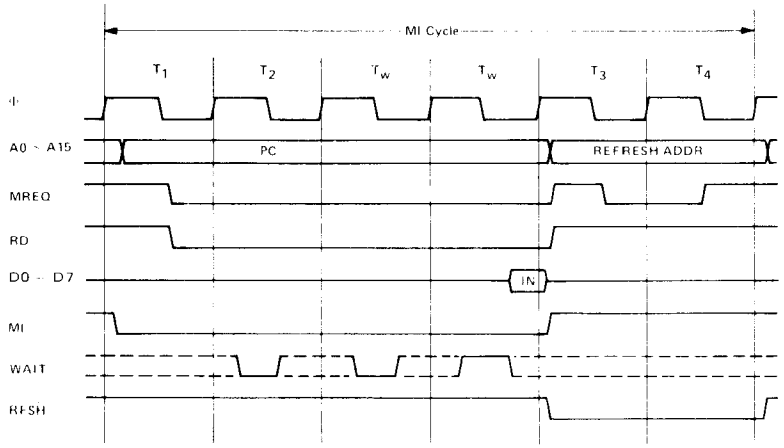


FIGURE 4.0-1A

MEMORY READ OR WRITE

Figure 4.0-2 illustrates the timing of memory read or write cycles other than an OP code fetch (M1 cycle). These cycles are generally three clock periods long unless wait states are requested by the memory via the $\overline{WAIT}$ signal. The $\overline{MREQ}$ signal and the $\overline{RD}$ signal are used the same as in the fetch cycle. In the case of a memory write cycle, the $\overline{MREQ}$ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The $\overline{WR}$ line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore the $\overline{WR}$ signal goes inactive one half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.

MEMORY READ OR WRITE CYCLES

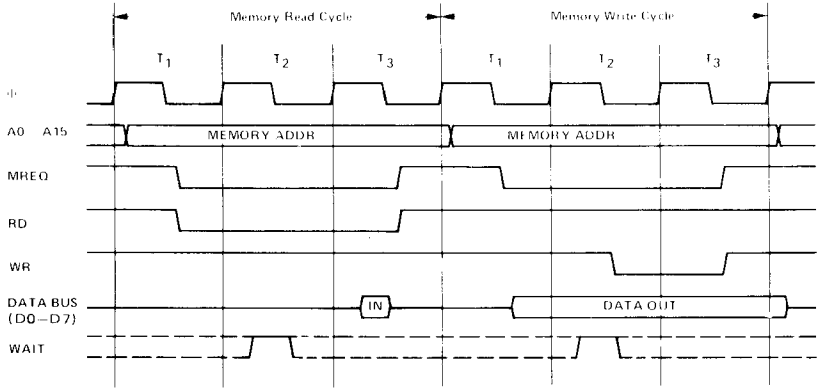


FIGURE 4.0-2

memory activates the $\overline{WAIT}$
e $\overline{WAIT}$ line with the falling
state will be entered during
be lengthened to match the

Figure 4.0-2A illustrates how a $\overline{WAIT}$ request signal will lengthen any memory read or write operation. This operation is identical to that previously described for a fetch cycle. Notice in this figure that a separate read and a separate write cycle are shown in the same figure although read and write cycles can never occur simultaneously.

MEMORY READ OR WRITE CYCLES WITH WAIT STATES

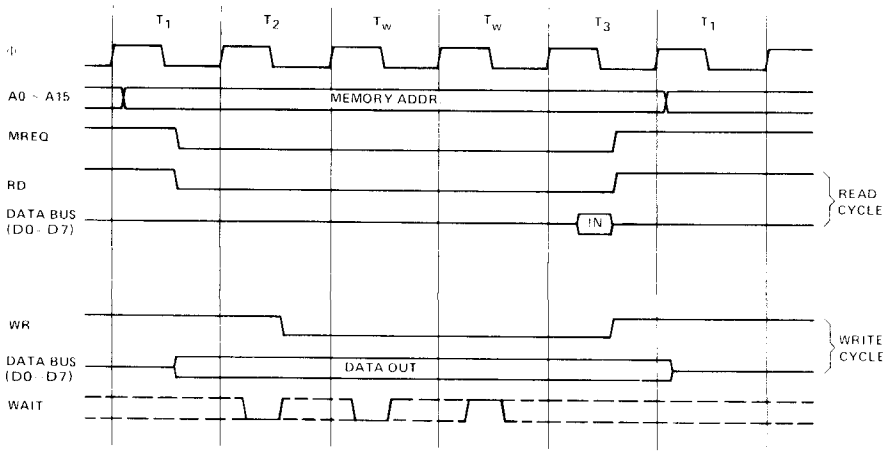


FIGURE 4.0-2A

INPUT OR OUTPUT CYCLES

Figure 4.0-3 illustrates an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted. The reason for this is that during I/O operations, the time from when the $\overline{IORQ}$ signal goes active until the CPU must sample the $\overline{WAIT}$ line is very short and without this extra state sufficient time does not exist for an I/O port to decode its address and activate the $\overline{WAIT}$ line if a wait is required. Also, without this wait state it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state time the $\overline{WAIT}$ request signal is sampled. During a read I/O operation, the $\overline{RD}$ line is used to enable the addressed port onto the data bus just as in the case of a memory read. For I/O write operations, the $\overline{WR}$ line is used as a clock to the I/O port, again with sufficient overlap timing automatically provided so that the rising edge may be used as a data clock.

Figure 4.0-3A illustrates how additional wait states may be added with the $\overline{WAIT}$ line. The operation is identical to that previously described.

BUS REQUEST/ACKNOWLEDGE CYCLE

Figure 4.0-4 illustrates the timing for a Bus Request/Acknowledge cycle. The $\overline{BUSRQ}$ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the $\overline{BUSRQ}$ signal is active, the CPU will set its address, data and tri-state control signals to the high impedance state with the rising edge of the next clock pulse. At that time any external device can control the buses to transfer data between memory and I/O devices. (This is generally known as Direct Memory Access [DMA] using cycle stealing). The maximum time for the CPU to respond to a bus request is the length of a machine cycle and the external controller can maintain control of the bus for as many clock cycles as is desired. Note, however, that if very long DMA cycles are used, and dynamic memories are being used, the external controller must also perform the refresh function. This situation only occurs if very large blocks of data are transferred under DMA control. Also note that during a bus request cycle, the CPU cannot be interrupted by either a $\overline{NMI}$ or an $\overline{INT}$ signal.

INPUT OR OUTPUT CYCLE

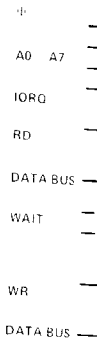


FIGURE 4.0-3

INPUT OR OUTPUT CYCLES W

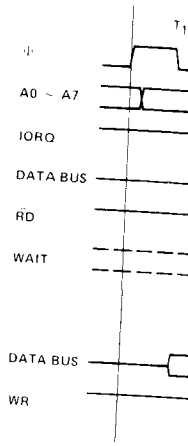
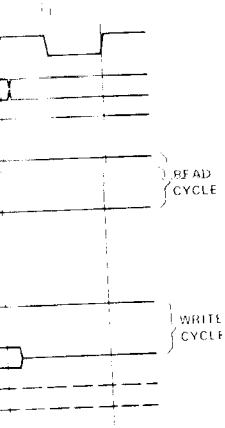


FIGURE 4.0-3A

any memory read or
bed for a fetch cycle.
are shown in the same

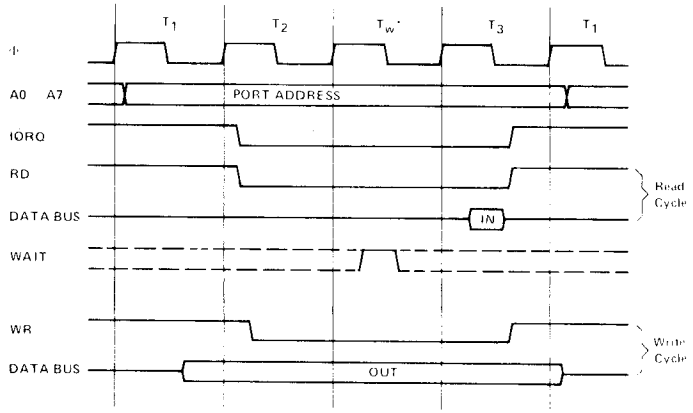


hat during I/O operations
hat during I/O operations,
ust sample the $\overline{WAIT}$ line
t exist for an I/O port to
d. Also, without this wait
at full CPU speed. During
a read I/O operation, the
s just as in the case of a
lock to the I/O port, again
rising edge may be used as

dded with the $\overline{WAIT}$ line.

wledge cycle. The $\overline{BUSRQ}$
ock period of any machine
ss, data and tri-state control
e next clock pulse. At that
a between memory and I/O
DMA] using cycle stealing).
t is the length of a machine
us for as many clock cycles
used, and dynamic memories
fresh function. This situation
DMA control. Also note that
ither a $\overline{NMI}$ or an $\overline{INT}$ signal.

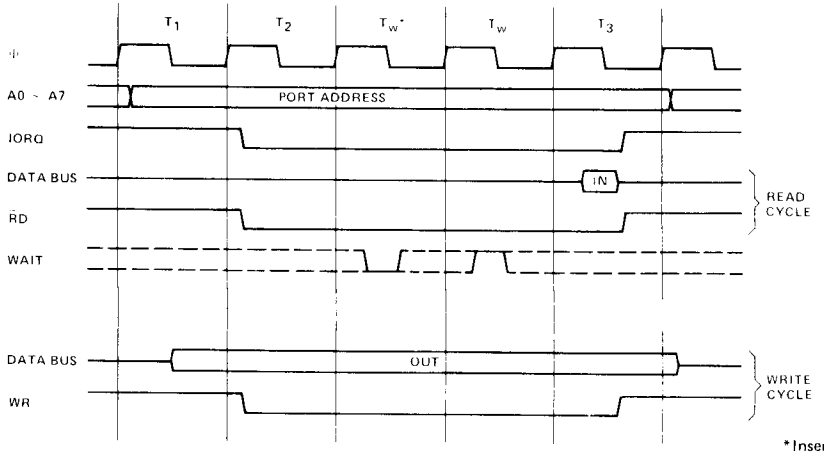
INPUT OR OUTPUT CYCLES



*Inserted by Z80 CPU

FIGURE 4.0-3

INPUT OR OUTPUT CYCLES WITH WAIT STATES



*Inserted by Z80 CPU

FIGURE 4.0-3A

BUS REQUEST/ACKNOWLEDGE CYCLE

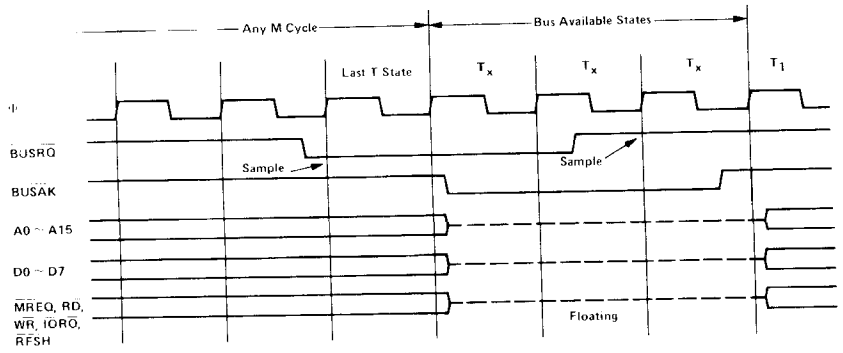


FIGURE 4.0-4

INTERRUPT REQUEST/ ACKNOWLEDGE CYCLE

Figure 4.0-5 illustrates the timing associated with an interrupt cycle. The interrupt signal ($\overline{INT}$) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the $\overline{BUSRQ}$ signal is active. When the signal is accepted a special M1 cycle is generated. During this special M1 cycle the $\overline{IORQ}$ signal becomes active (instead of the normal $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stablilize and identify which I/O device must insert the response vector. Refer to section 8.0 for details on how the interrupt response vector is utilized by the CPU.

INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

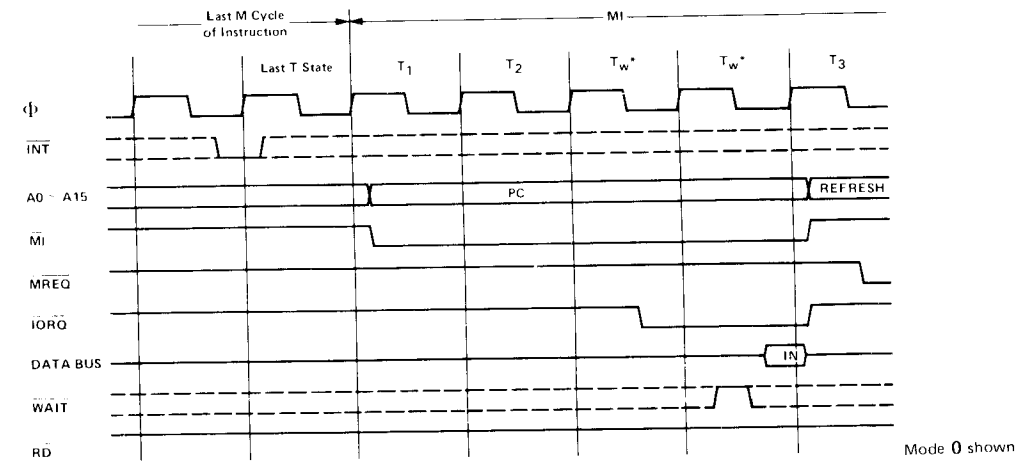


FIGURE 4.0-5

Figure 4.0-5A illustrat
cycle. Again the operat

INTERRUPT REQUEST/ACKN

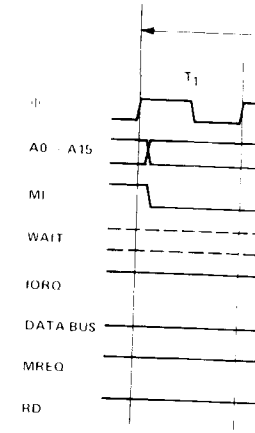


FIGURE 4.0-5A

NON MASKABLE INTE

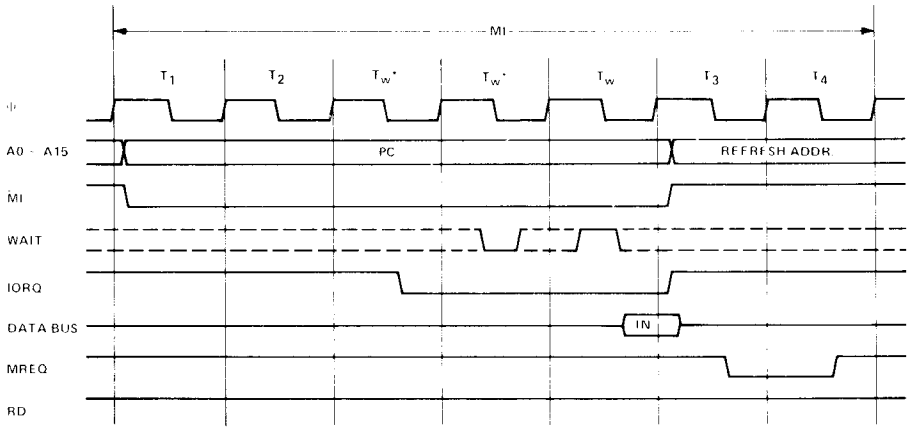
Figure 4.0-6 illustrates
A pulse on the $\overline{NMI}$ inp
end of every instruction,
but this line has priority
ware control. Its usual
such as an impending p
similar to a normal mem
of the data bus is ignore
stack and jumps to locat
must begin at this locatior

HALT EXIT

Whenever a software halt
interrupt is received (eith
flip flop is enabled). The
each T4 state as shown in
maskable interrupt has bee
state will be exited on the
rupt acknowledge cycle co
received at this time, then
priority. The purpose of e
memory refresh signals ac
except that the data receiv
internally to the CPU. Th
that the processor is in the

Figure 4.0-5A illustrates how additional wait states can be added to the interrupt response cycle. Again the operation is identical to that previously described.

INTERRUPT REQUEST/ACKNOWLEDGE WITH WAIT STATES



Mode **0** shown

FIGURE 4.0-5A

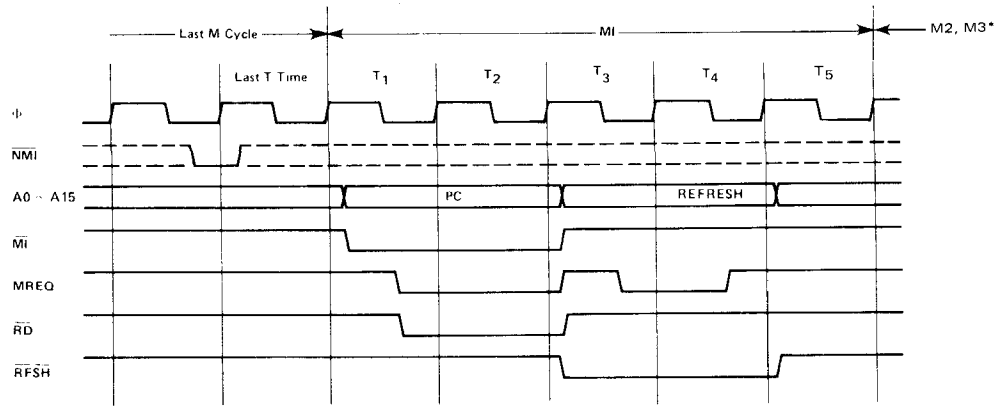
NON MASKABLE INTERRUPT RESPONSE

Figure 4.0-6 illustrates the request/acknowledge cycle for the non-maskable interrupt. A pulse on the $\overline{NMI}$ input sets an internal NMI latch which is tested by the CPU at the end of every instruction. This NMI latch is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt and it can not be disabled under software control. Its usual function is to provide immediate response to important signals such as an impending power failure. The CPU response to a non maskable interrupt is similar to a normal memory read operation. The only difference being that the content of the data bus is ignored while the processor automatically stores the PC in the external stack and jumps to location 0066_H. The service routine for the non maskable interrupt must begin at this location if this interrupt is used.

HALT EXIT

Whenever a software halt instruction is executed the CPU begins executing NOP's until an interrupt is received (either a non-maskable or a maskable interrupt while the interrupt flip flop is enabled). The two interrupt lines are sampled with the rising clock edge during each T4 state as shown in Figure 4.0-7. If a non-maskable interrupt has been received or a maskable interrupt has been received and the interrupt enable flip-flop is set, then the halt state will be exited on the next rising clock edge. The following cycle will then be an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the non maskable one will be acknowledged since it was highest priority. The purpose of executing NOP instructions while in the halt state is to keep the memory refresh signals active. Each cycle in the halt state is a normal M1 (fetch) cycle except that the data received from the memory is ignored and a NOP instruction is forced internally to the CPU. The halt acknowledge signal is active during this time to indicate that the processor is in the halt state.

NON MASKABLE INTERRUPT REQUEST OPERATION



*M2 and M3 are stack write operations

FIGURE 4.0-6

HALT EXIT

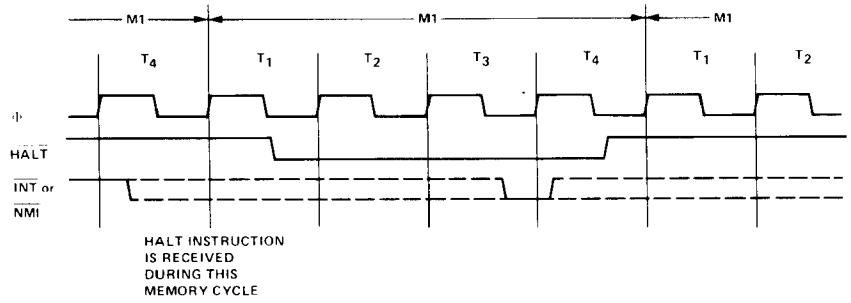


FIGURE 4.0-7

5.0 Z80-CPU INSTR

The Z80-CPU can ex
CPU. The instructions

- Load and Excha
- Block Transfer a
- Arithmetic and L
- Rotate and Shift
- Bit Manipulation
- Jump, Call and R
- Input/Output
- Basic CPU Contro

5.1 INTRODUCTION

The load instructions m
and external memory. A
the data is to be move
a load instruction. Exam
general purpose register
also includes load imm
Other types of load instru
The exchange instruction

A unique set of block tra
block of memory of any
block moves is extremely
block search instruction
instruction, a block of ex
character. Once the chara
block transfer and the bloc
as to not occupy the CPU

The arithmetic and logical
general purpose CPU regis
are placed in the accumul
the operation. An example
tents of an external mem
accumulator. This group al
registers.

The bit manipulation instru
register or any external mem
For example, the most sign
useful in control applicatio
ramming.

The jump, call and return in
user's program. This group u
counter address from specifi
restart instruction. This instr
OP code. This is possible since
memory may be specified. Pr
or IY directly into the PC, th
routine being executed.

of tes	No.of M Cycles	No.of T States	Comments
	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	4 (If B = 0)	16	
	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	4 (If B = 0)	16	
	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	4 (If B = 0)	16	
	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	5 (If B ≠ 0)	21	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
	4 (If B = 0)	16	

8.0 INTERRUPT RESPONSE

The prupose of an interrupt is to allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start a peripheral service routine. Usually this service routine is involved with the exchange of data, or status and control information, between the CPU and the peripheral. Once the service routine is completed, the CPU returns to the operation from which it was interrupted.

INTERRUPT ENABLE – DISABLE

The Z80-CPU has two interrupt inputs, a software maskable interrupt and a non-maskable interrupt. The non-maskable interrupt ($\overline{NMI}$) can not be disabled by the programmer and it will be accepted whenever a peripheral device requests it. This interrupt is generally reserved for very important functions that must be serviced whenever they occur, such as an impending power failure. The maskable interrupt ($\overline{INT}$) can be selectively enabled or disabled by the programmer. This allows the programmer to disable the interrupt during periods where his program has timing constraints that do not allow it to be interrupted. In the Z80-CPU there is an enable flip flop (called IFF) that is set or reset by the programmer using the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. When the IFF is reset, an interrupt can not be accepted by the CPU.

Actually, for purposes that will be subsequently explained, there are two enable flip flops, called IFF_1 and IFF_2.

$\boxed{IFF_1}$	$\boxed{IFF_2}$
Actually disables interrupts from being accepted.	Temporary storage location for IFF_1.

The state of IFF_1 is used to actually inhibit interrupts while IFF_2 is used as a temporary storage location for IFF_1. The purpose of storing the IFF_1 will be subsequently explained.

A reset to the CPU will force both IFF_1 and IFF_2 to the reset state so that interrupts are disabled. They can then be enabled by an EI instruction at any time by the programmer. When an EI instruction is executed, any pending interrupt request will not be accepted until after the instruction following EI has been executed. This single instruction delay is necessary for cases when the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instructions sets both IFF_1 and IFF_2 to the enable state. When an interrupt is accepted by the CPU, both IFF_1 and IFF_2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF_1 and IFF_2 are always equal.

The purpose of IFF_2 is to save the status of IFF_1 when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF_1 is reset to prevent further interrupts until reenabled by the programmer. Thus, after a non-maskable interrupt has been accepted maskable interrupts are disabled but the previous state of IFF_1 has been saved so that the complete state of the CPU just prior to the non-maskable interrupt can be restored at any time. When a Load Register A with Register I (LD A, I) instruction or a Load Register A with Register R (LD A, R) instruction is executed, the state of IFF_2 is copied into the parity flag where it can be tested or stored.

A second method of restoring the status of IFF_1 is thru the execution of a Return From Non-Maskable Interrupt (RETN) instruction. Since this instruction indicates that the non maskable interrupt service routine is complete, the contents of IFF_2 are now copied back into IFF_1, so that the status of IFF_1 just prior to the acceptance of the non-maskable interrupt will be restored automatically.

Figure 8.0-1 is a summary of the effect of different instructions on the two enable flip flops.

INTERRUPT ENABLE/DISABLE FLIP FLOPS

Action	IFF_1	IFF_2	
CPU Reset	0	0	
DI	0	0	
EI	1	1	
LD A, I	•	•	$IFF_2 \rightarrow$ Parity flag
LD A, R	•	•	$IFF_2 \rightarrow$ Parity flag
Accept NMI	0	•	
RETN	IFF_2	•	$IFF_2 \rightarrow IFF_1$
Accept INT	0	0	
RETI	•	•	

"•" indicates no change

FIGURE 8.0-1

CPU RESPONSE

Non-Maskable

A non-maskable interrupt will be accepted at all times by the CPU. When this occurs, the CPU ignores the next instruction that it fetches and instead does a restart to location 0066H. Thus, it behaves exactly as if it had received a restart instruction but, it is to a location that is not one of the 8 software restart locations. A restart is merely a call to a specific address in page 0 memory.

Maskable

The CPU can be programmed to respond to the maskable interrupt in any one of three possible modes.

Mode 0

This mode is identical to the 8080A interrupt response mode. With this mode, the interrupting device can place any instruction on the data bus and the CPU will execute it. Thus, the interrupting device provides the next instruction to be executed instead of the memory. Often this will be a restart instruction since the interrupting device only need supply a single byte instruction. Alternatively, any other instruction such as a 3 byte call to any location in memory could be executed.

The number of clock cycles necessary to execute this instruction is 2 more than the normal number for the instruction. This occurs since the CPU automatically adds 2 wait states to an interrupt response cycle to allow sufficient time to implement an external daisy chain for priority control. Section 4.0 illustrates the detailed timing for an interrupt response. After the application of RESET the CPU will automatically enter interrupt Mode 0.

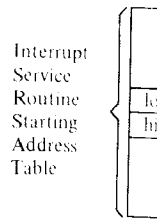
Mode 1

When this mode has been selected by the programmer, the CPU will respond to an interrupt by executing a restart to location 0038H. Thus the response is identical to that for a non maskable interrupt except that the call location is 0038H instead of 0066H. Another difference is that the number of cycles required to complete the restart instruction is 2 more than normal due to the two added wait states.

Mode 2

This mode is the most powerf
user an indirect call can be mac

With this mode the programm
terrupt service routine. This ta
is accepted, a 16 bit pointer n
starting address from the table
of the I register. The I register
the programmer, i.e. LD I, A,
tialized to zero. The lower eig
device. Actually, only 7 bits
bit must be a zero. This is rec
from a complete 16 bit service
in even locations.



The first byte in the table is t
programmer must obviously fill
are to be accepted.

Note that this table can be char
Write Memory) to allow differe

Once the interrupting device sup
cally pushes the program counte
and does a jump to this address
plete (7 to fetch the lower 8
counter, and 6 to obtain the jum

Note that the Z80 peripheral de
that automatically supplies the
ledge. Refer to the Z80-PIO, Z80

on the two enable flip flops.

→Parity flag

→Parity flag

→IFF1

CPU. When this occurs, the
d does a restart to location
rt instruction but, it is to a
restart is merely a call to a

nterrupt in any one of three

With this mode, the interrupt-
CPU will execute it. Thus, the
uted instead of the memory.
g device only need supply a
ch as a 3 byte call to any lo-

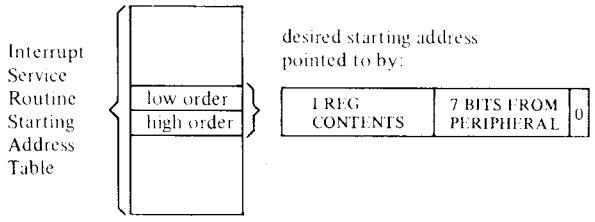
on is 2 more than the normal
ically adds 2 wait states to an
t an external daisy chain for
an interrupt response. After
errupt Mode 0.

U will respond to an interrupt
is identical to that for a non
instead of 0066H. Another
e the restart instruction is 2

Mode 2

This mode is the most powerful interrupt response mode. With a single 8-bit byte from the user an indirect call can be made to any memory location.

With this mode the programmer maintains a table of 16 bit starting addresses for every interrupt service routine. This table may be located anywhere in memory. When an interrupt is accepted, a 16 bit pointer must be formed to obtain the desired interrupt service routine starting address from the table. The upper 8 bits of this pointer is formed from the contents of the I register. The I register must have been previously loaded with the desired value by the programmer, i.e. LD I, A. Note that a CPU reset clears the I register so that it is initialized to zero. The lower eight bits of the pointer must be supplied by the interrupting device. Actually, only 7 bits are required from the interrupting device as the least bit must be a zero. This is required since the pointer is used to get two adjacent bytes to from a complete 16 bit service routine starting address and the addresses must always start in even locations.



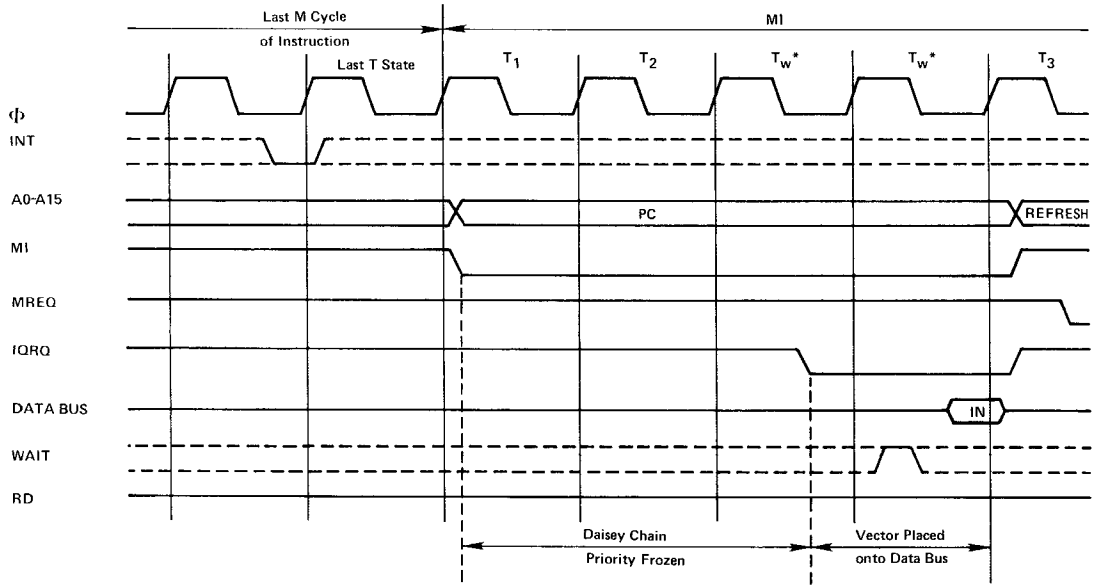
The first byte in the table is the least significant (low order) portion of the address. The programmer must obviously fill this table in with the desired addresses before any interrupts are to be accepted.

Note that this table can be changed at any time by the programmer (if it is stored in Read/Write Memory) to allow different peripherals to be serviced by different service routines.

Once the interrupting device supplies the lower portion of the pointer, the CPU automatically pushes the program counter onto the stack, obtains the starting address from the table and does a jump to this address. This mode of response requires 19 clock periods to complete (7 to fetch the lower 8 bits from the interrupting device, 6 to save the program counter, and 6 to obtain the jump address.)

Note that the Z80 peripheral devices all include a daisy chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80-PIO, Z80-SIO and Z80-CTC manuals for details.

INTERRUPT REQUEST/ACKNOWLEDGE CYCLE



Z80 INTERRUPT ACKNOWLEDGE SUMMARY

1) PERIPHERAL DEVICE REQUESTS INTERRUPT. Any device requesting and interrupt can pull the wired-or line $\overline{INT}$ low.

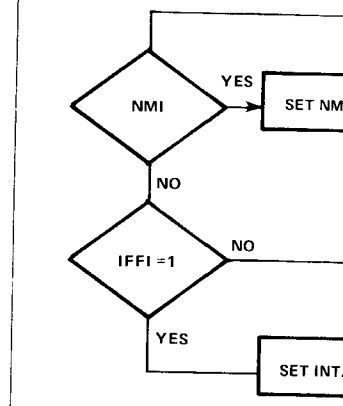
2) CPU ACKNOWLEDGES INTERRUPT. Priority status is frozen when $\overline{M1}$ goes low during the Interrupt Acknowledge sequence. Propagation delays down the IEI/IEO daisy chain must be settled out when $\overline{IORQ}$ goes low. If IEI is HIGH, an active Peripheral Device will place its Interrupt Vector on the Data Bus when $\overline{IORQ}$ goes low. That Peripheral then releases its hold on $\overline{INT}$ allowing interrupts from a higher priority device. Lower priority devices are inhibited from placing their Vector on the Data Bus or Interrupting because IEO is low on the active device.

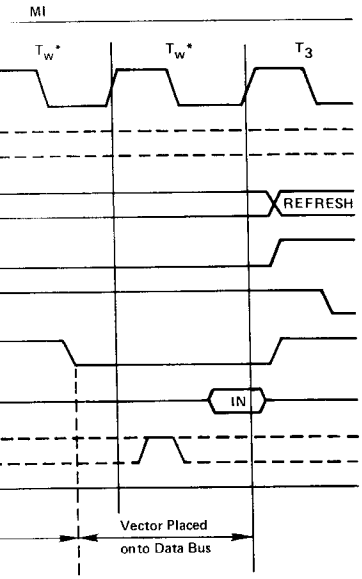
3) INTERRUPT IS CLEARED. An active Peripheral device (IEI=1, IEO=0) monitors OP Code fetches for an RETI (ED 4D) instruction which tells the peripheral that its Interrupt Service Routine is over. The peripheral device then re-activates its internal Interrupt structure as well as raising its IEO line to enable lower priority devices.

INTERRELATIONS

The following flow
the following from t
1. $\overline{INT}$ and $\overline{NMI}$ are
2. $\overline{BUSRQ}$ is acted o
3. While the CPU is i
4. These three input

Z80-CPU INTERRUPT SEQUE





vice requesting and interrupt

frozen when $\overline{M1}$ goes low
delays down the IEI/IEO
IEI is HIGH, an active Peri-
Bus when $\overline{IORQ}$ goes low.
upts from a higher
om placing their Vector on
e device.

e (IEI=1, IEO=0) monitors
tells the peripheral that its
then re-activates its internal
wer priority devices.

INTERRELATIONSHIP OF $\overline{INT}$, $\overline{NMI}$, AND $\overline{BUSRQ}$

The following flow chart details the relationship of three control inputs to the Z80-CPU. Note the following from the flow chart.

1. $\overline{INT}$ and $\overline{NMI}$ are always acted on at the end of an instruction.
2. $\overline{BUSRQ}$ is acted on at the end of a machine cycle.
3. While the CPU is in the DMA MODE, it will not respond to active inputs on $\overline{INT}$ or $\overline{NMI}$.
4. These three inputs are acted on in the following order of priority: a)$\overline{BUSRQ}$ b)$\overline{NMI}$ c)$\overline{INT}$