

Mostek

Z80 PROCESSOR

Technical Manual

MK3880

Central Processing Unit

1979 MICROCOMPUTER
COMPONENT DATA BOOK

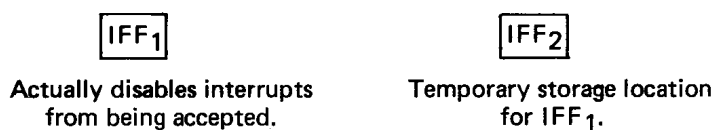
8.0 INTERRUPT RESPONSE

The purpose 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 ($\overline{\text{NMI}}$) 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 ($\overline{\text{INT}}$) 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 IFF₁ and IFF₂.



The state of IFF₁ is used to actually inhibit interrupts while IFF₂ is used as a temporary storage location for IFF₁. The purpose of storing the IFF₁ will be subsequently explained.

A reset to the CPU will force both IFF₁ and IFF₂ 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 instruction sets both IFF₁ and IFF₂ to the enable state. When an interrupt is accepted by the CPU, both IFF₁ and IFF₂ are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF₁ and IFF₂ are always equal.

The purpose of IFF₂ is to save the status of IFF₁ when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF₁ 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 IFF₁ 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 IFF₂ is copied into the parity flag where it can be tested or stored.

A second method of restoring the status of IFF₁ 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 IFF₂ are now copied back into IFF₁, so that the status of IFF₁ 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 → Parity flag
LD A, R	•	•	IFF ₂ → Parity flag
Accept NMI	0	•	
RETN	IFF ₂	•	IFF ₂ → 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.

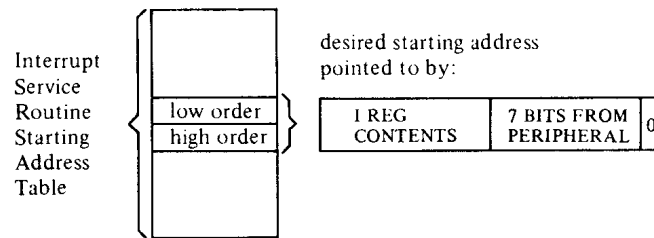
Mode 1

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 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 form a complete 16 bit service routine starting address and the addresses must always start in even locations.



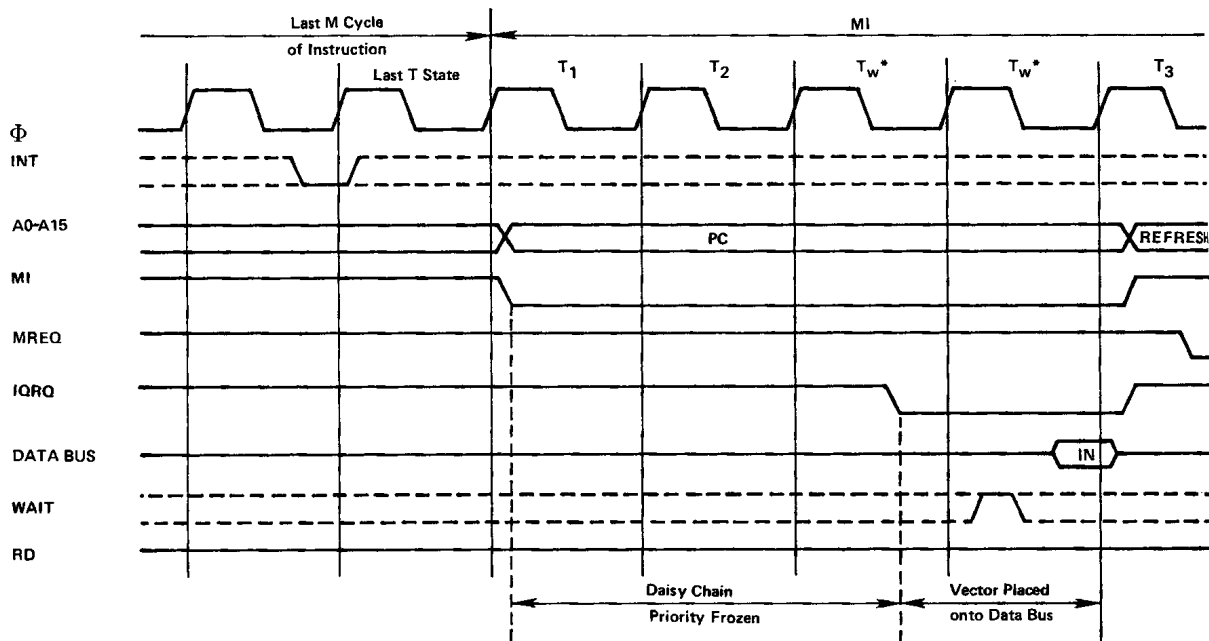
The first byte in the table is the least significant (low order) portion of the address. The programmer must obviously fill this table in with the desired addresses before any interrupts are to be accepted.

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.

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

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

1. $\overline{\text{INT}}$ and $\overline{\text{NMI}}$ are always acted on at the end of an instruction.
2. $\overline{\text{BUSRQ}}$ 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 $\overline{\text{INT}}$ or $\overline{\text{NMI}}$.
4. These three inputs are acted on in the following order of priority: a) $\overline{\text{BUSRQ}}$ b) $\overline{\text{NMI}}$ c) $\overline{\text{INT}}$

