Architectural Overview

Zilog's Z80 CPU family of components are fourth-generation enhanced microprocessors with exceptional computational power. They offer higher system throughput and more efficient memory utilization than comparable second and third-generation microprocessors. The speed offerings from 6–20 MHz suit a wide range of applications which migrate software. The internal registers contain 208 bits of read/write memory that are accessible to the programmer. These registers include two sets of six general-purpose registers which can be used individually as either 8-bit registers or as 16-bit register pairs. In addition, there are two sets of Accumulator and Flag registers.

The Z80 CPU also contains a Stack Pointer, Program Counter, two index registers, a refresh register, and an interrupt register. The CPU is easy to incorporate into a system because it requires only a single +5V power source. All output signals are fully decoded and timed to control standard memory or peripheral circuits; the Z80 CPU is supported by an extensive family of peripheral controllers.

Figure 1 shows the internal architecture and major elements of the Z80 CPU.

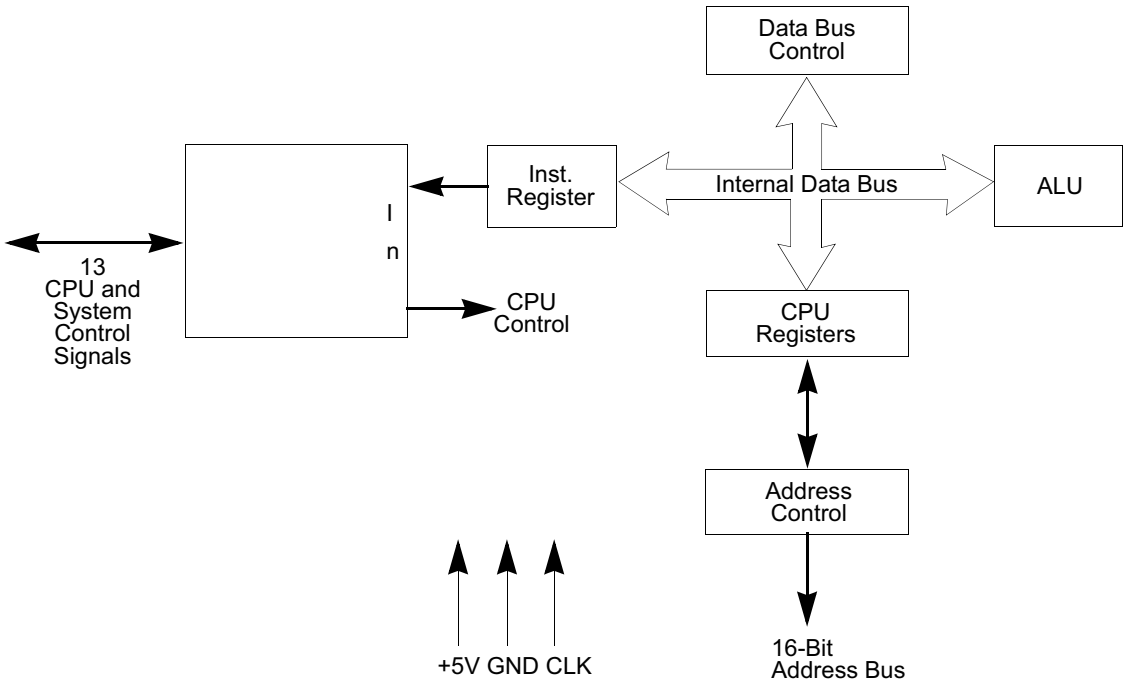

Figure 1. Z80 CPU Block Diagram

CPU Register

The Z80 CPU contains 208 bits of read/write memory that are available to the programmer. Figure 2 shows how this memory is configured to eighteen 8-bit registers and four 16-bit registers. All Z80 CPU's registers are implemented using static RAM. The registers include two sets of six general-purpose registers that can be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of Accumulator and Flag registers and six special-purpose registers.

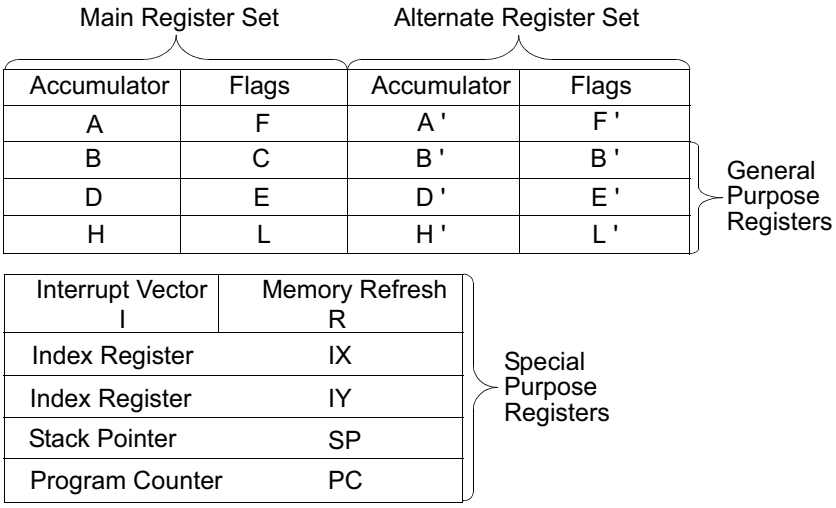

Figure 2. CPU Register Configuration

Special-Purpose Registers

Program Counter (PC). The program counter holds the 16-bit address of the current instruction being fetched from memory. The Program Counter is automatically incremented after its contents are transferred to the address lines. When a program jump occurs, the new value is automatically placed in the Program Counter, overriding the incrementer.

Stack Pointer (SP). The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack to specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the most recent data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

Two Index Registers (IX and IY). The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially when tables of data are used.

Interrupt Page Address (I) Register. The Z80 CPU can be operated in a mode in which an indirect call to any memory location can be achieved in response to an interrupt. The I register is used for this purpose and stores the high-order eight bits of the indirect address while the interrupting device provides the lower eight bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with minimal access time to the routine.

Memory Refresh (R) Register. The Z80 CPU contains a memory refresh counter, enabling dynamic memories to be used with the same ease as static memories. Seven bits of this 8-bit register are automatically incremented after each instruction fetch. The eighth bit remains as programmed, resulting from an LD *R*, *A* instruction. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is transparent to the programmer and does not slow the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer. During refresh, the contents of the I Register are placed on the upper eight bits of the address bus.

Accumulator and Flag Registers. The CPU includes two independent 8-bit Accumulators and associated 8-bit Flag registers. The Accumulator holds the results of 8-bit arithmetic or logical operations while the Flag Register indicates specific conditions for 8-bit or 16-bit operations, such as indicating whether or not the result of an operation is equal to 0. The programmer selects the Accumulator and flag pair with a single exchange instruction so that it is possible to work with either pair.

General Purpose Registers

Two matched sets of general-purpose registers, each set containing six 8-bit registers, can be used individually as 8-bit registers or as 16-bit register pairs. One set is called *BC*, *DE*, and *HL* while the complementary set is called *BC'*, *DE'*, and *HL'*. At any one time, the programmer can select either set of registers to work through a single exchange command for the entire set. In systems that require fast interrupt response, one set of general-purpose registers and an Accumulator/Flag Register can be reserved for handling this fast routine. One exchange command is executed to switch routines. This process greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general-purpose reg-

isters are used for a wide range of applications. They also simplify programing, specifically in ROM-based systems in which little external read/write memory is available.

Arithmetic Logic Unit

The 8-bit arithmetic and logical instructions of the CPU are executed in the Arithmetic Logic Unit (ALU). Internally, the ALU communicates with the registers and the external data bus by using the internal data bus. Functions performed by the ALU include:

- Add
- Subtract
- Logical AND
- Logical OR
- Logical exclusive OR
- Compare
- Left or right shifts or rotates (arithmetic and logical)
- Increment
- Decrement
- Set bit
- Reset bit
- Test bit

Instruction Register and CPU Control

As each instruction is fetched from memory, it is placed in the Instruction Register and decoded. The control sections performs this function and then generates and supplies the control signals necessary to read or write data from or to the registers, control the ALU, and provide required external control signals.

Pin Description

The Z80 CPU I/O pins are shown in Figure 3. The function of each pin is described in the section that follows.

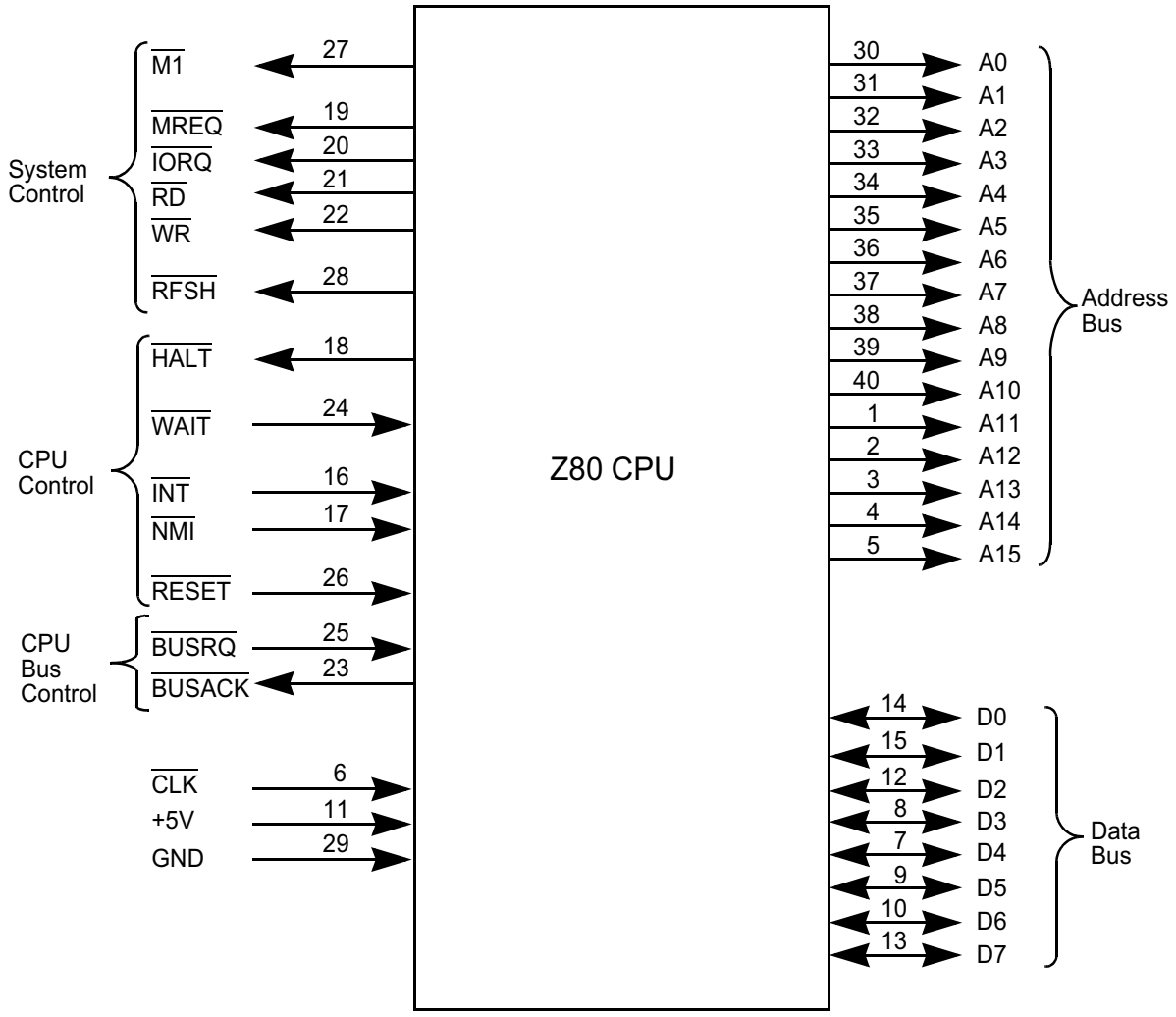

Figure 3. Z80 CPU I/O Pin Configuration

Pin Functions

A15–A0. *Address Bus (output, active High, tristate).* A15–A0 form a 16-bit Address Bus, which provides the addresses for memory data bus exchanges (up to 64KB) and for I/O device exchanges.

BUSACK. *Bus Acknowledge (output, active Low).* Bus Acknowledge indicates to the requesting device that the CPU address bus, data bus, and control signals $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$,

$\overline{RD}$, and $\overline{WR}$ have entered their high-impedance states. The external circuitry can now control these lines.

BUSREQ. *Bus Request (input, active Low).* Bus Request contains a higher priority than $\overline{NMI}$ and is always recognized at the end of the current machine cycle. $\overline{BUSREQ}$ forces the CPU address bus, data bus, and control signals $\overline{MREQ}$, $\overline{IORQ}$, RD, and WR to enter a high-impedance state so that other devices can control these lines. $\overline{BUSREQ}$ is normally *wired OR* and requires an external pull-up for these applications. Extended $\overline{BUSREQ}$ periods due to extensive DMA operations can prevent the CPU from properly refreshing dynamic RAM.

D7–D0. *Data Bus (input/output, active High, tristate).* D7–D0 constitute an 8-bit bidirectional data bus, used for data exchanges with memory and I/O.

HALT. *HALT State (output, active Low).* $\overline{HALT}$ indicates that the CPU has executed a HALT instruction and is waiting for either a nonmaskable or a maskable interrupt (with the mask enabled) before operation can resume. During HALT, the CPU executes NOPs to maintain memory refreshes.

INT. *Interrupt Request (input, active Low).* An Interrupt Request is generated by I/O devices. The CPU honors a request at the end of the current instruction if the internal software-controlled interrupt enable flip-flop (IFF) is enabled. $\overline{INT}$ is normally wired-OR and requires an external pull-up for these applications.

IORQ. *Input/Output Request (output, active Low, tristate).* $\overline{IORQ}$ indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. $\overline{IORQ}$ is also generated concurrently with $\overline{M1}$ during an interrupt acknowledge cycle to indicate that an interrupt response vector can be placed on the data bus.

M1. *Machine Cycle One* (output, active Low). $\overline{M1}$, together with $\overline{MREQ}$, indicates that the current machine cycle is the op code fetch cycle of an instruction execution. $\overline{M1}$, when operating together with $\overline{IORQ}$, indicates an interrupt acknowledge cycle.

MREQ. *Memory Request (output, active Low, tristate).* $\overline{MREQ}$ indicates that the address bus holds a valid address for a memory read or a memory write operation.

NMI. *Nonmaskable Interrupt (input, negative edge-triggered).* $\overline{NMI}$ contains a higher priority than $\overline{INT}$. $\overline{NMI}$ is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop, and automatically forces the CPU to restart at location `0066h`.

RD. *Read (output, active Low, tristate).* $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

RESET. *Reset (input, active Low).* $\overline{RESET}$ initializes the CPU as follows: it resets the interrupt enable flip-flop, clears the Program Counter and registers I and R, and sets the interrupt status to Mode 0. During reset time, the address and data bus enter a high-impedance state, and all control output signals enter an inactive state. $\overline{RESET}$ must be active for a minimum of three full clock cycles before a reset operation is complete.

RFSH. *Refresh (output, active Low).* $\overline{\text{RFSH}}$, together with $\overline{\text{MREQ}}$, indicates that the lower seven bits of the system's address bus can be used as a refresh address to the system's dynamic memories.

WAIT. *WAIT (input, active Low).* $\overline{\text{WAIT}}$ communicates to the CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter a $\overline{\text{WAIT}}$ state as long as this signal is active. Extended $\overline{\text{WAIT}}$ periods can prevent the CPU from properly refreshing dynamic memory.

WR. *Write (output, active Low, tristate).* $\overline{\text{WR}}$ indicates that the CPU data bus contains valid data to be stored at the addressed memory or I/O location.

CLK. *Clock (input).* Single-phase MOS-level clock.

Note: All signals with an overline are active Low. For example, B/$\overline{\text{W}}$, in which *word* is active Low, or $\overline{\text{B}}$/W, in which *byte* is active Low.

Timing

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

- Memory read or write
- I/O device read or write
- Interrupt acknowledge

All instructions are a series of basic operations. Each of these 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. These clock periods are referred to as time (T) cycles, and the operations are referred to as machine (M) cycles. Figure 4 shows how a typical instruction is a series of specific M and T cycles. In Figure 4, this instruction consists of the three machine cycles M1, M2, and M3. The first machine cycle of any instruction is a fetch cycle that is four, five, or six T cycles long (unless lengthened by the WAIT signal, which is 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 can feature anywhere from three to five T cycles (again, they can be lengthened by wait states to synchronize external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles.

During T2 and every subsequent automatic WAIT state (TW), the CPU samples the WAIT line with the falling edge of the clock. If the WAIT line is active at this time, another

WAIT state is entered during the following cycle. Using this technique, the read can be lengthened to match the access time of any type of memory device. See the Input or Output Cycles section on page 10 to learn more about the automatic WAIT state.

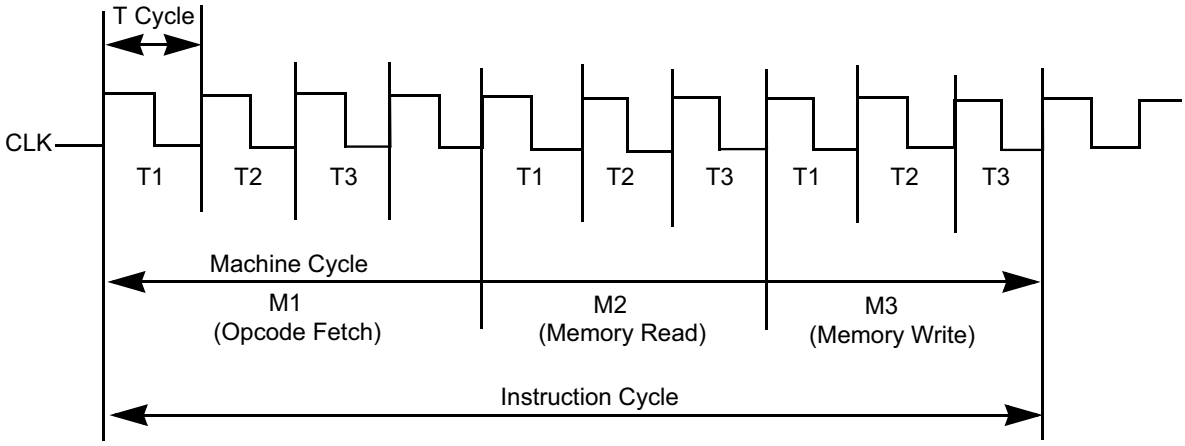

Figure 4. Basic CPU Timing Example

Instruction Fetch

Figure 5 depicts the timing during an M1 (op code fetch) cycle. The Program Counter is placed on the address bus at the beginning of the M1 cycle. One half clock cycle later, the $\overline{MREQ}$ signal goes active. At this time, the address to 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 space 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. As a result, the data is sampled by the CPU before the $\overline{RD}$ signal becomes inactive. Clock states 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 concurrent operation can be performed.

During T3 and T4, the lower seven bits of the address bus contain a memory refresh address and the $\overline{RFSH}$ signal becomes active, indicating that a refresh read of all dynamic memories must be performed. To prevent data from different memory segments from being gated onto the data bus, an $\overline{RD}$ signal is not generated during this refresh period. The $\overline{MREQ}$ signal during this refresh period should be used to perform a refresh read of all memory elements. The refresh signal cannot be used by itself, because the refresh address is only guaranteed to be stable during the $\overline{MREQ}$ period.

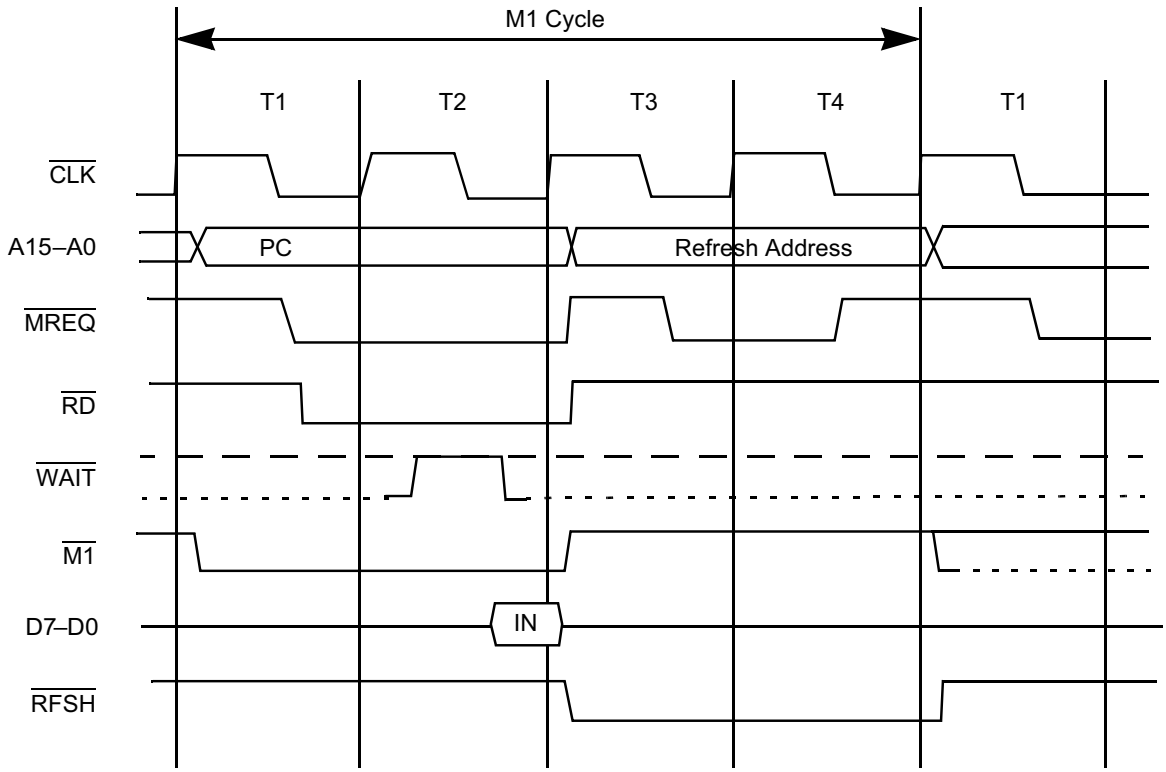

Figure 5. Instruction Op Code Fetch

Memory Read Or Write

Figure 6 shows the timing of memory read or write cycles other than an op code fetch cycle. These cycles are generally three clock periods long unless wait states are requested by memory through the $\overline{WAIT}$ signal. The $\overline{MREQ}$ signal and the $\overline{RD}$ signal are used the same way as in a fetch cycle. In 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 the 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 almost any type of semiconductor memory type is met.

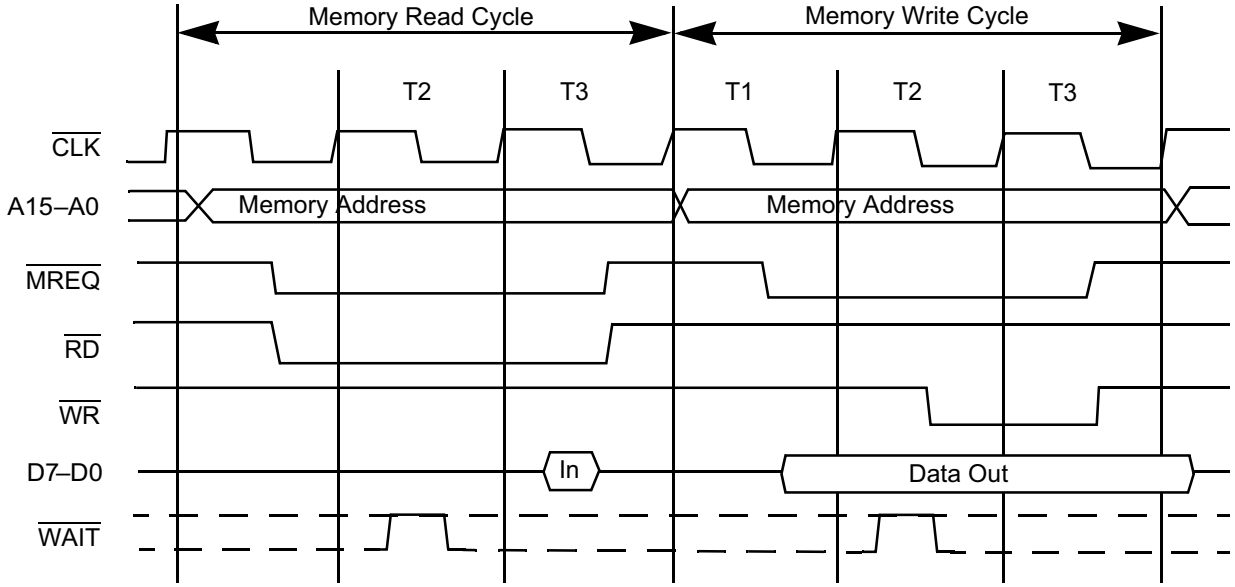

Figure 6. Memory Read or Write Cycle

Input or Output Cycles

Figure 7 shows an I/O read or I/O write operation. During I/O operations, a single wait state is automatically inserted. The reason for this single wait state insertion is that during I/O operations, the period from when the $\overline{\text{IORQ}}$ signal goes active until the CPU must sample the $\overline{\text{WAIT}}$ line is short. Without this extra state, sufficient time does not exist for an I/O port to decode its address and activate the $\overline{\text{WAIT}}$ line if a wait is required. Additionally, without this wait state, it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state period, the $\overline{\text{WAIT}}$ request signal is sampled.

During a read I/O operation, the $\overline{\text{RD}}$ line is used to enable the addressed port onto the data bus, just as in the case of a memory read. The $\overline{\text{WR}}$ line is used as a clock to the I/O port for write operations.

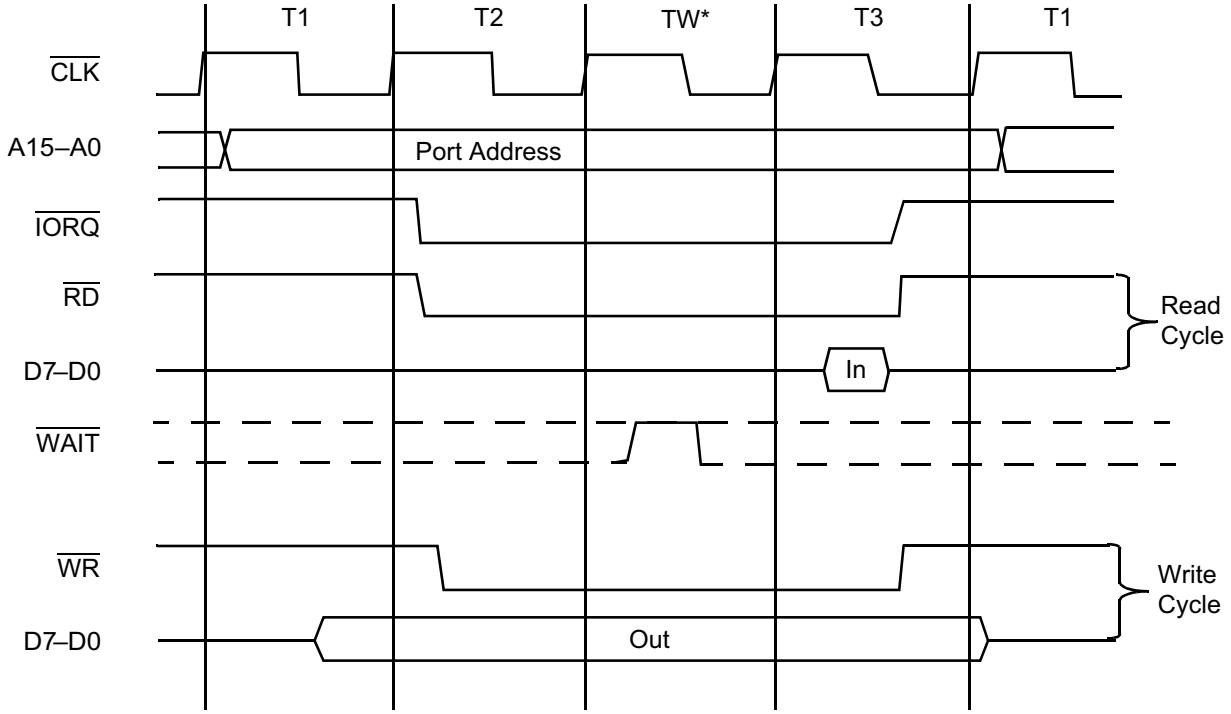

Figure 7. Input or Output Cycles

Note: *In Figure 7, TW is an automatically-inserted WAIT state.

Bus Request/Acknowledge Cycle

Figure 8 shows the timing for a Bus Request/Acknowledge cycle. The $\overline{\text{BUSREQ}}$ signal is sampled by the CPU with the rising edge of the most recent clock period of any machine cycle. If the $\overline{\text{BUSREQ}}$ signal is active, the CPU sets its address, data, and tristate 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 operation 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 required. If long DMA cycles are used, and dynamic memories are used, the external controller also performs the refresh function. This situation only occurs if

large blocks of data are transferred under DMA control. During a bus request cycle, the CPU cannot be interrupted by either an $\overline{NMI}$ or an $\overline{INT}$ signal.

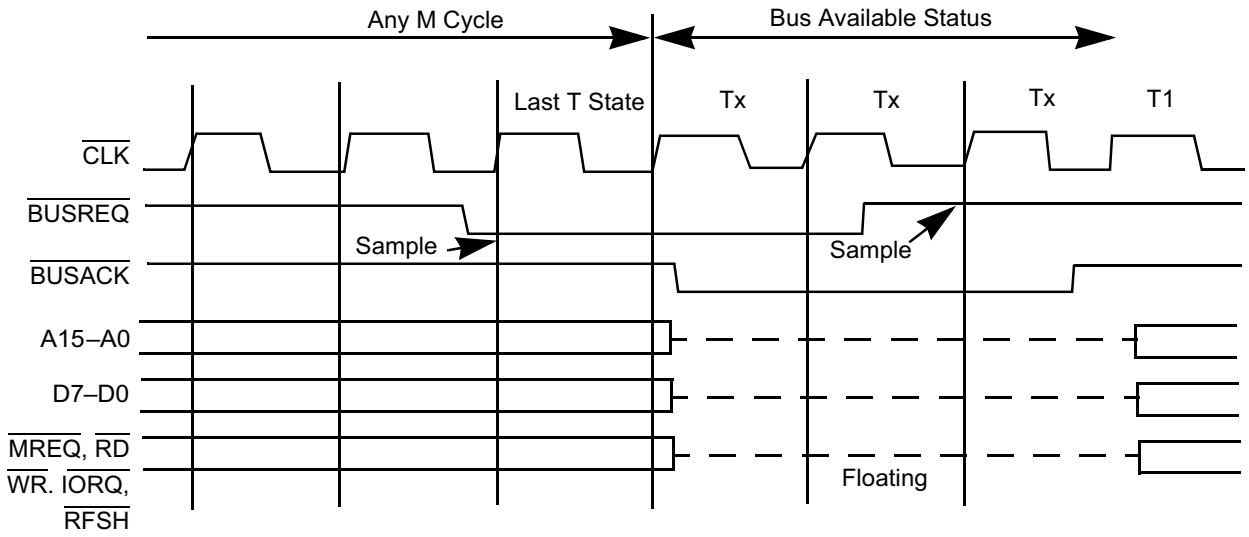

Figure 8. Bus Request/Acknowledge Cycle

Interrupt Request/Acknowledge Cycle

Figure 9 shows the timing associated with an interrupt cycle. The CPU samples the interrupt signal ($\overline{INT}$) with the rising edge of the final clock at the end of any instruction. The signal is not accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the $\overline{BUSREQ}$ 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. 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 stabilize and identify which I/O device must insert the response vector. Refer to the Interrupt Response section on page 17 to learn more about how the interrupt response vector is utilized by the CPU.

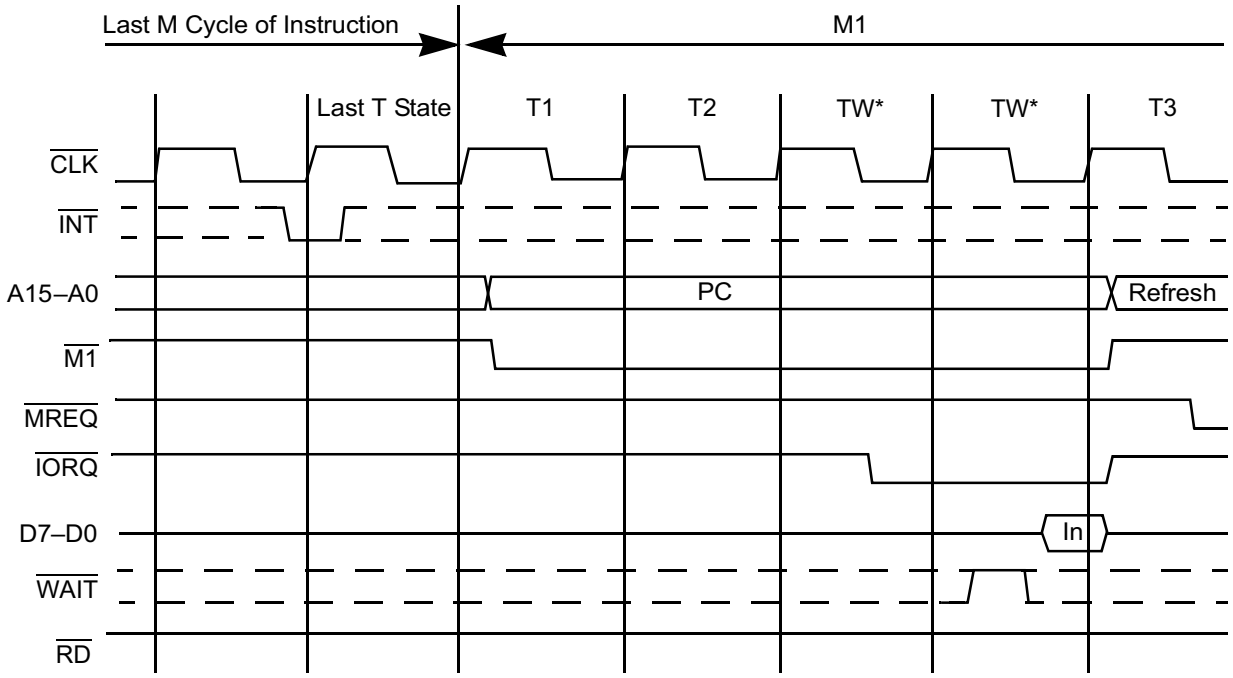

Figure 9. Interrupt Request/Acknowledge Cycle

Nonmaskable Interrupt Response

Figure 10 shows the request/acknowledge cycle for the nonmaskable interrupt. This signal is sampled at the same time as the interrupt line, but this line takes priority over the normal interrupt and it cannot 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 nonmaskable interrupt is similar to a normal memory read operation. The only difference is that the contents of the data bus are ignored while the processor automatically stores the Program Counter in the external stack and jumps to address `0066h`. The service routine for the nonmaskable interrupt must begin at this location if this interrupt is used.

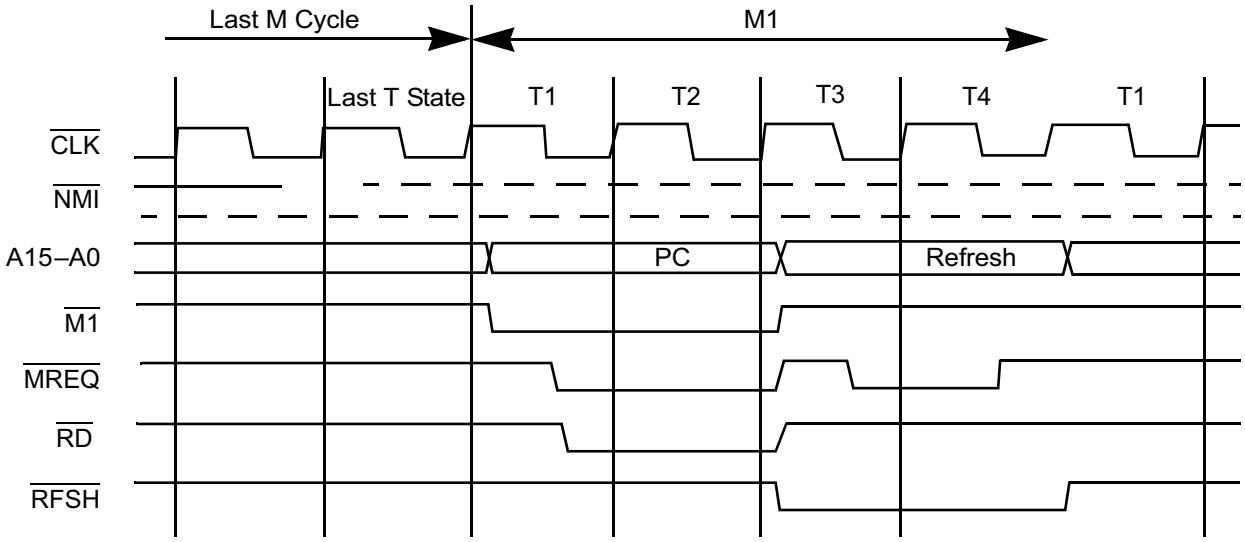

Figure 10. Nonmaskable Interrupt Request Operation

HALT Exit

When a software HALT instruction is executed, the CPU executes NOPs until an interrupt is received (either a nonmaskable 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 depicted in Figure 11. If a nonmaskable interrupt is received or a maskable interrupt is received and the interrupt enable flip-flop is set, then the HALT state is exited on the next rising clock edge. The following cycle is an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the nonmaskable interrupt is acknowledged because it is the 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 an NOP instruction is forced internally to the CPU. The HALT acknowledge signal is active during this time indicating that the processor is in the HALT state.

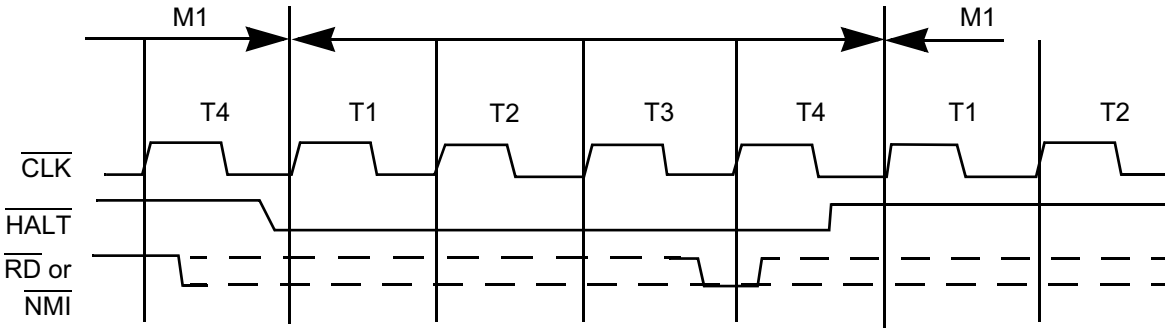

Figure 11. HALT Exit

Note: The HALT instruction is repeated during the memory cycle shown in Figure 11.

Power-Down Acknowledge Cycle

When the clock input to the Z80 CPU is stopped at either a High or Low level, the Z80 CPU stops its operation and maintains all registers and control signals. However, ICC2 (standby supply current) is guaranteed only when the system clock is stopped at a Low level during T4 of the machine cycle following the execution of the HALT instruction. The timing diagram for the power-down function (when implemented with the HALT instruction) is shown in Figure 12.

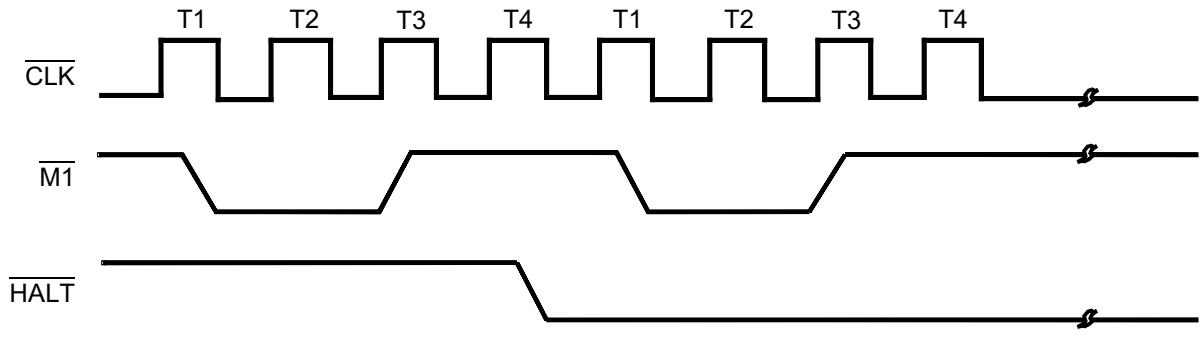

Figure 12. Power-Down Acknowledge

Power-Down Release Cycle

The system clock must be supplied to the Z80 CPU to release the power-down state. When the system clock is supplied to the CLK input, the Z80 CPU restarts operations from the point at which the power-down state was implemented. The timing diagrams for the release from power-down mode are featured in Figures 13 through 15. When the HALT instruction is executed to enter the power-down state, the Z80 CPU also enters the HALT state. An interrupt signal (either $\overline{\text{NMI}}$ or ANT) or a $\overline{\text{RESET}}$ signal must be applied to the CPU after the system clock is supplied to release the power-down state.

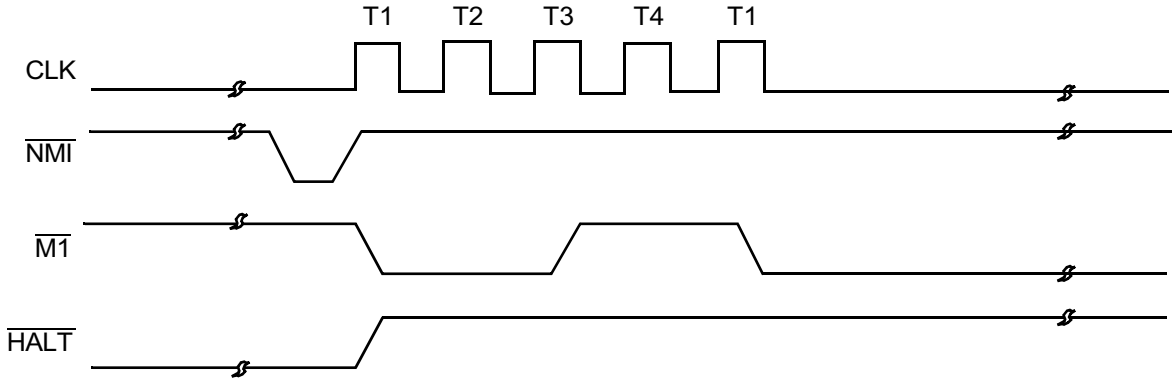

Figure 13. Power-Down Release Cycle, #1 of 3

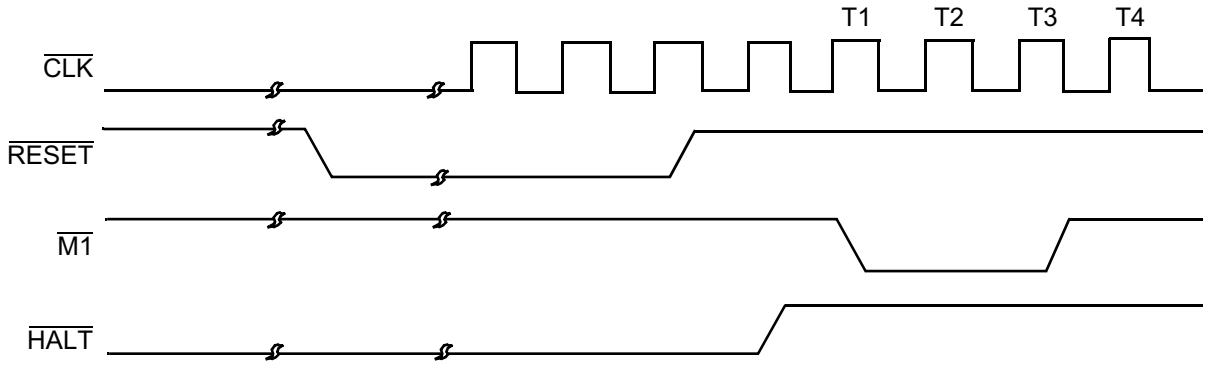

Figure 14. Power-Down Release Cycle, #2 of 3

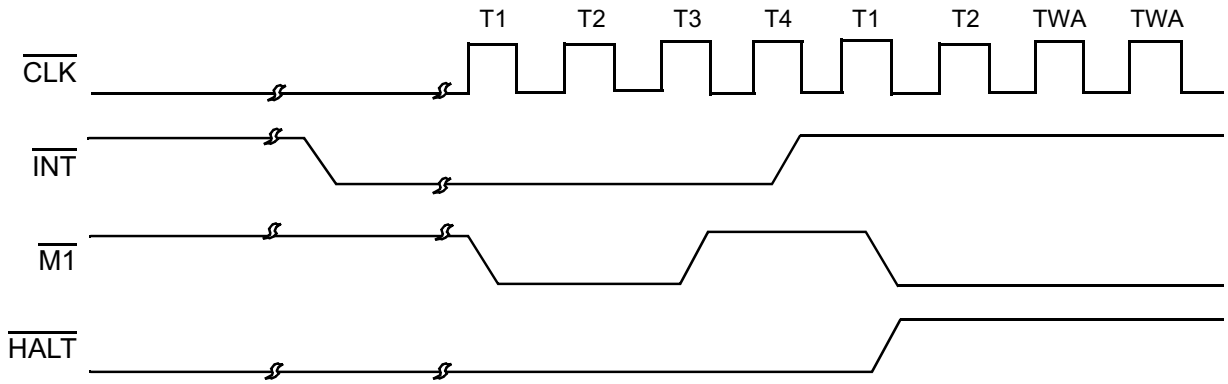

Figure 15. Power-Down Release Cycle, #3 of 3

Interrupt Response

An interrupt allows peripheral devices to suspend CPU operation and force the CPU to start a peripheral service routine. This service routine usually involves the exchange of data, status, or control information between the CPU and the peripheral. When the service routine is completed, the CPU returns to the operation from which it was interrupted.

Interrupt Enable/Disable

The Z80 CPU contains two interrupt inputs: a software maskable interrupt ($\overline{INT}$) and a nonmaskable interrupt ($\overline{NMI}$). The nonmaskable interrupt cannot be disabled by the programmer and is accepted when a peripheral device requests it. This interrupt is generally reserved for important functions that can be enabled or disabled selectively by the programmer. This routine allows the programmer to disable the interrupt during periods when the program contains timing constraints that wont allow interrupt. In the Z80 CPU, there is an interrupt enable flip-flop (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 cannot be accepted by the CPU.

The two enable flip-flops are IFF1 and IFF2, as depicted in Figure 16.

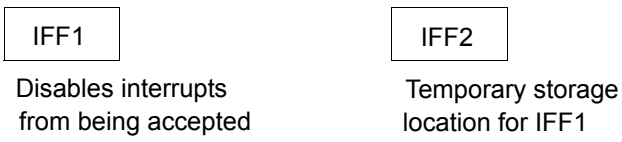

Figure 16. Interrupt Enable Flip-Flops

The state of IFF1 is used to inhibit interrupts while IFF2 is used as a temporary storage location for IFF1.

A CPU reset forces both the IFF1 and IFF2 to the reset state, which disables interrupts. Interrupts can be enabled at any time by an EI instruction from the programmer. When an EI instruction is executed, any pending interrupt request is not accepted until after the instruction following EI is executed. This single instruction delay is necessary when the next instruction is a return instruction. Interrupts are not allowed until a return is completed. The EI instruction sets both IFF1 and IFF2 to the enable state. When the CPU accepts a maskable interrupt, both IFF1 and IFF2 are automatically reset, inhibiting further interrupts until the programmer issues a new El instruction.

Note: 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 nonmaskable interrupt occurs. When a nonmaskable interrupt is accepted, IFF1 resets to prevent further interrupts until reenabled by the programmer. Therefore, after a nonmaskable interrupt is accepted, maskable interrupts are disabled but the previous state of IFF1 is saved so that the complete state of the CPU just prior to the nonmaskable 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 to the parity flag, where it can be tested or stored.

A second method of restoring the status of IFF1 is through the execution of a Return From Nonmaskable Interrupt (RETN) instruction. This instruction indicates that the nonmaskable interrupt service routine is complete and the contents of IFF2 are now copied back into IFF1 so that the status of IFF1 just prior to the acceptance of the nonmaskable interrupt is restored automatically.

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

Table 1. Interrupt Enable/Disable, Flip-Flops

Action	IFF1	IFF2	Comments
CPU Reset	0	0	Maskable interrupt, $\overline{INT}$ disabled.
DI Instruction Execution	0	0	Maskable $\overline{INT}$ disabled.
EI Instruction Execution	1	1	Maskable, $\overline{INT}$ enabled.
LD A,I Instruction Execution	*	*	IFF2 → Parity flag.
LD A,R instruction Execution	*	*	IFF2 → Parity flag.

Table 1. Interrupt Enable/Disable, Flip-Flops (Continued)

Action	IFF1	IFF2	Comments
Accept $\overline{\text{NMI}}$	0	*	Maskable → Interrupt.
RETN Instruction Execution	IFF2	*	IFF2 → Indicates completion of nonmaskable interrupt service routine.

CPU Response

The CPU always accepts a nonmaskable interrupt. When this nonmaskable interrupt is accepted, the CPU ignores the next instruction that it fetches and instead performs a restart at address 0066h. The CPU functions as if it had recycled a restart instruction, but to a location other than one of the eight software restart locations. A restart is merely a call to a specific address in Page 0 of memory.

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

Mode 0

Mode 0 is similar to the 8080A interrupt response mode. With Mode 0, the interrupting device can place any instruction on the data bus and the CPU executes it. Consequently, the interrupting device provides the next instruction to be executed. Often this response is a restart instruction because the interrupting device is required to supply only 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 two more than the normal number for the instruction. The addition of two clock cycles occurs because the CPU automatically adds two wait states to an Interrupt response cycle to allow sufficient time to implement an external daisy-chain for priority control. Figures 9 and 10 on page 13 show the timing for an interrupt response. After the application of $\overline{\text{RESET}}$, the CPU automatically enters interrupt Mode 0.

Mode 1

When Mode 1 is selected by the programmer, the CPU responds to an interrupt by executing a restart at address 0038h. As a result, the response is identical to that of a nonmaskable interrupt except that the call location is 0038h instead of 0066h. The number of cycles required to complete the restart instruction is two more than normal due to the two added wait states.

Mode 2

Mode 2 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.

In Mode 2, the programmer maintains a table of 16-bit starting addresses for every interrupt service routine. This table can be located anywhere in memory. When an interrupt is accepted, a 16-bit pointer must be formed to obtain the required interrupt service routine starting address from the table. The upper eight bits of this pointer is formed from the contents of the I Register. The I register must be loaded with the applicable value by the programmer, such as LD *I*, *A*. A CPU reset clears the I Register so that it is initialized to 0. The lower eight bits of the pointer must be supplied by the interrupting device. Only seven bits are required from the interrupting device, because the least-significant bit must be a 0. This process is required, because the pointer must receive two adjacent bytes to form a complete 16-bit service routine starting address; addresses must always start in even locations.

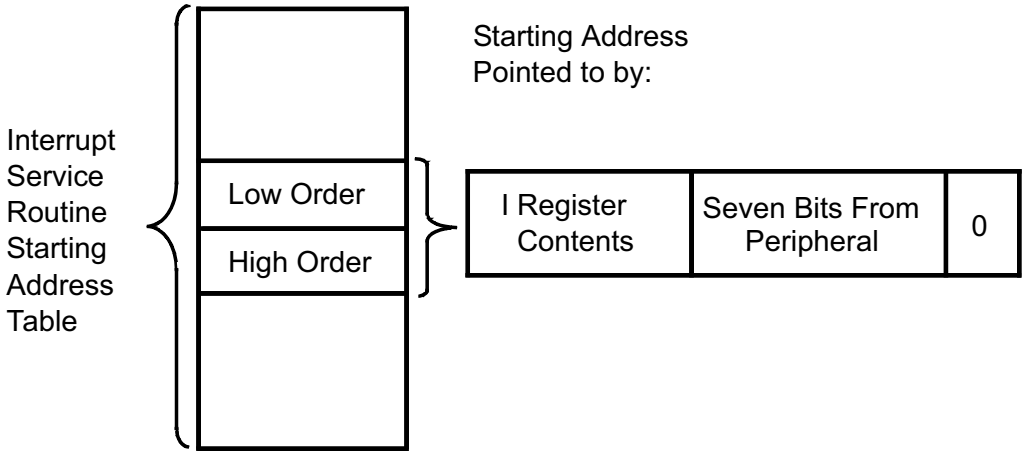

Figure 17. Mode 2 Interrupt Response Mode

The first byte in the table is the least-significant (low-order portion of the address). The programmer must complete the table with the correct addresses before any interrupts are accepted.

The programmer can change the table by storing it in read/write memory, which also allows individual peripherals to be serviced by different service routines.

When 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 performs a jump to this address. This mode of response requires 19 clock periods to complete (seven to fetch the lower eight bits from the interrupting device, six to save the program counter, and six to obtain the jump address).

The Z80 peripheral devices include a daisy-chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80 CPU Peripherals User Manual (UM0081) for more complete information.