Z80 Microprocessors

Z80 CPU

User Manual

UM008006-0714

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer

Revision History

Each instance in the following revision history table reflects a change to this document from its previous version. For more details, refer to the corresponding pages provided in the table.

Date	Revision Level	Description	Page
Jul 2014	06	Updated to Zilog style and to incorporate customer suggestions, including a correction to the Z80 Status Indicator Flags table, bit 4, and a correction to the EXX instruction at bit 0.	63, 124
Feb 2005	05	Corrected the hex code for the RLCA instruction; corrected illustration for the Rotate and Shift Group RLCA instruction.	53, 203
Dec 2004	04	Corrected discrepancies in the bit patterns for IM 0, IM 1 and IM 2 instructions.	182, 183, 184

Table of Contents

List of Figures

List of Tables

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.

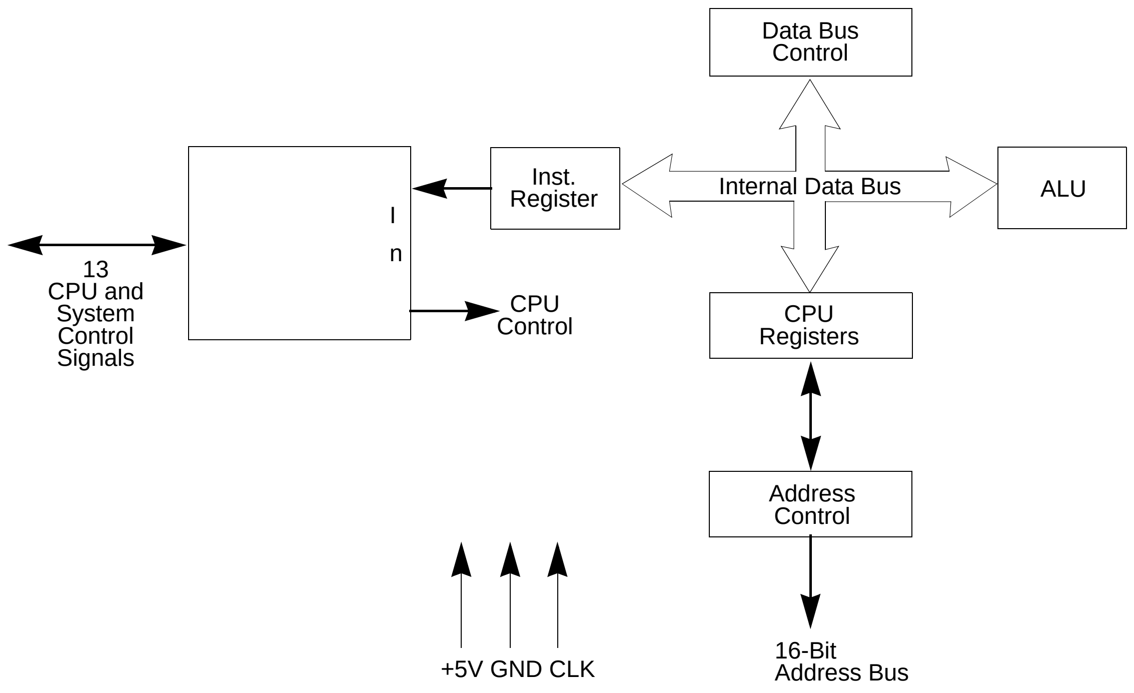


Figure 1. Z80 CPU Block Diagram

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.

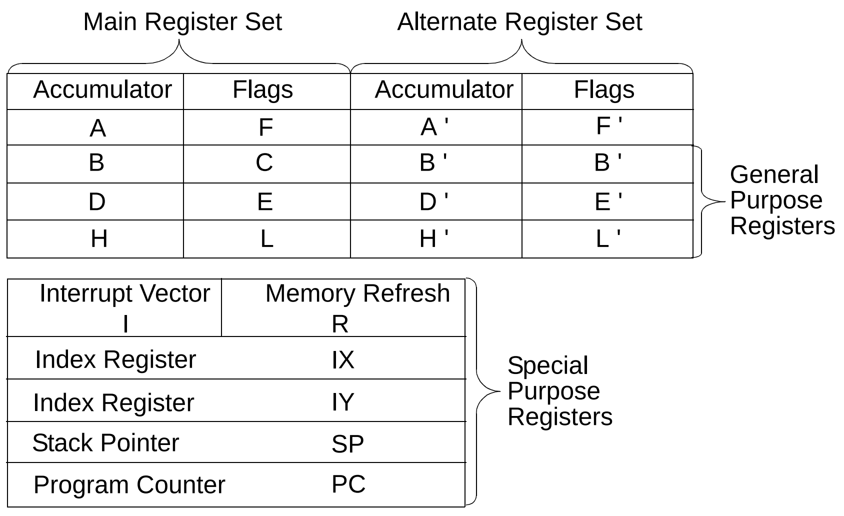


Figure 2. CPU Register Configuration

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 registers 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.

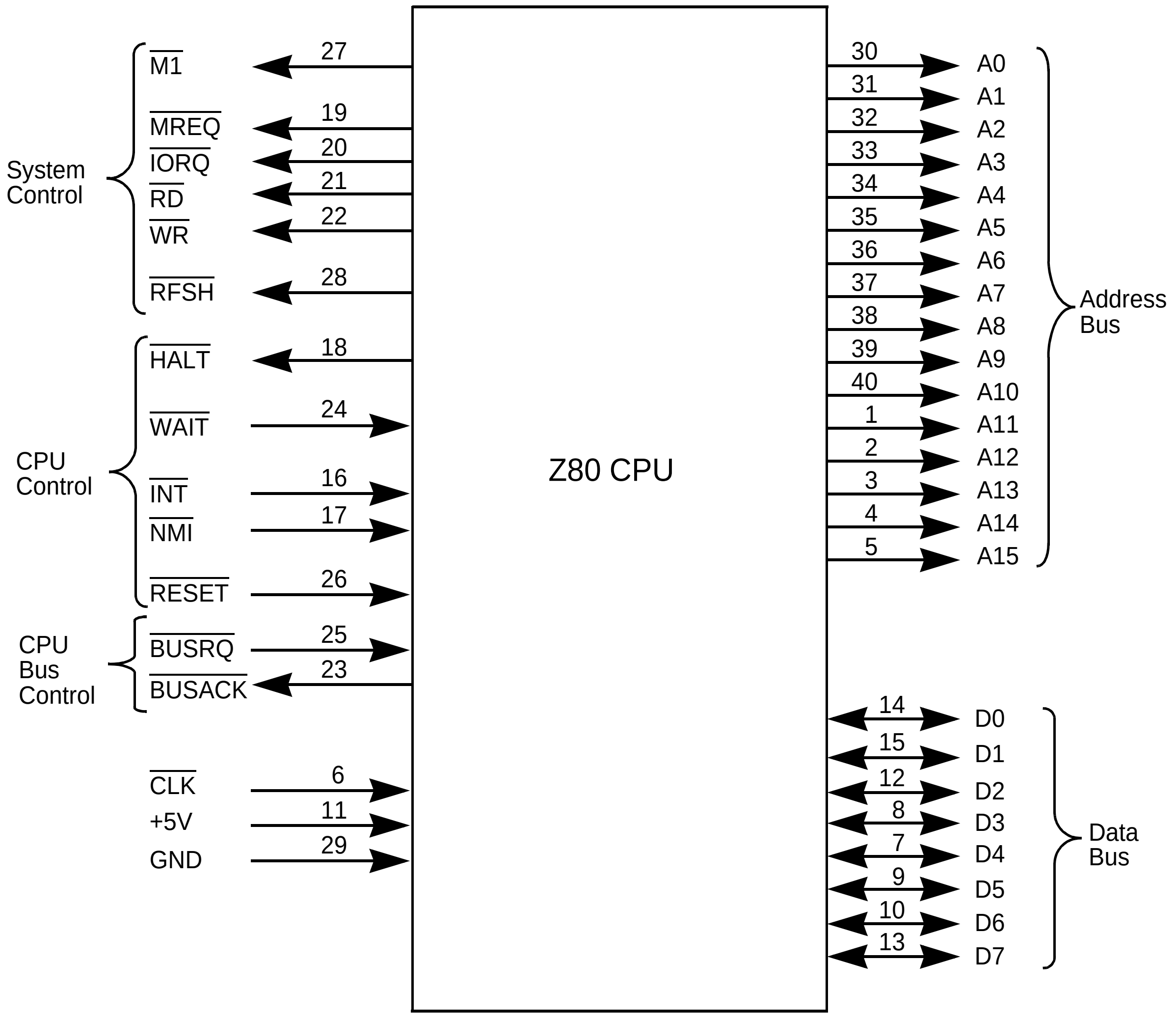


Figure 3. Z80 CPU I/O Pin Configuration

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 $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$,

$\overline{\text{RD}}$, and $\overline{\text{WR}}$ 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 $\overline{\text{NMI}}$ and is always recognized at the end of the current machine cycle. $\overline{\text{BUSREQ}}$ forces the CPU address bus, data bus, and control signals $\overline{\text{MREQ}}$, $\overline{\text{IORQ}}$, $\overline{\text{RD}}$, and WR to enter a high-impedance state so that other devices can control these lines. $\overline{\text{BUSREQ}}$ is normally *wired OR* and requires an external pull-up for these applications. Extended $\overline{\text{BUSREQ}}$ 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).* $\overline{\text{HALT}}$ 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. $\overline{\text{INT}}$ is normally wired-OR and requires an external pull-up for these applications.

IORQ. *Input/Output Request (output, active Low, tristate).* $\overline{\text{IORQ}}$ indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. $\overline{\text{IORQ}}$ is also generated concurrently with $\overline{\text{M1}}$ 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). $\overline{\text{M1}}$, together with $\overline{\text{MREQ}}$, indicates that the current machine cycle is the op code fetch cycle of an instruction execution. $\overline{\text{M1}}$, when operating together with $\overline{\text{IORQ}}$, indicates an interrupt acknowledge cycle.

MREQ. *Memory Request (output, active Low, tristate).* $\overline{\text{MREQ}}$ 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).* $\overline{\text{NMI}}$ contains a higher priority than $\overline{\text{INT}}$. $\overline{\text{NMI}}$ 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).* $\overline{\text{RD}}$ 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).* $\overline{\text{RESET}}$ 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. $\overline{\text{RESET}}$ 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 $\overline{\text{overline}}$ 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.

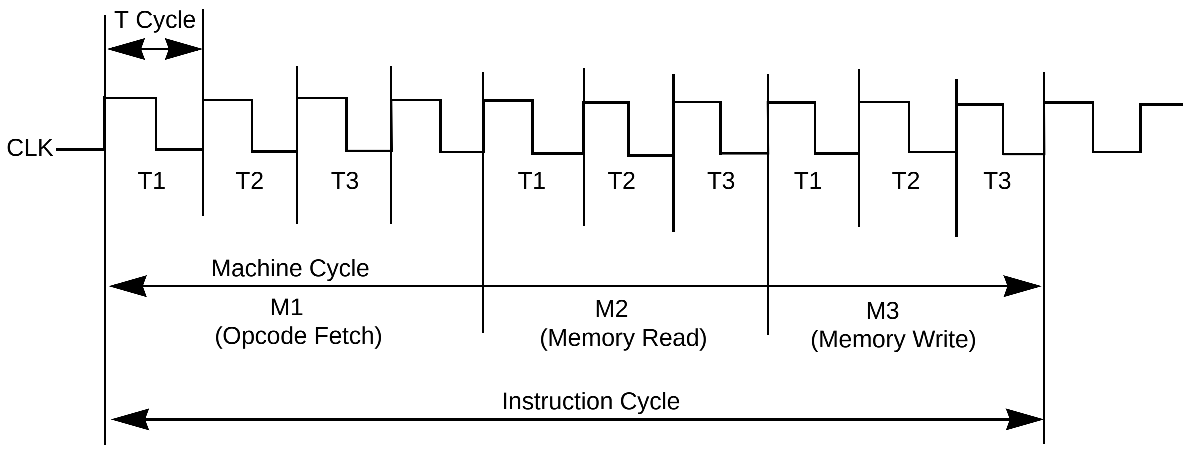


Figure 4. Basic CPU Timing Example

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 $\overline{\text{MREQ}}$ signal goes active. At this time, the address to memory has had time to stabilize so that the falling edge of $\overline{\text{MREQ}}$ can be used directly as a chip enable clock to dynamic memories. The $\overline{\text{RD}}$ 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 $\overline{\text{RD}}$ and $\overline{\text{MREQ}}$ signals. As a result, the data is sampled by the CPU before the $\overline{\text{RD}}$ 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 $\overline{\text{RFSH}}$ 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 $\overline{\text{RD}}$ signal is not generated during this refresh period. The $\overline{\text{MREQ}}$ 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 $\overline{\text{MREQ}}$ period.

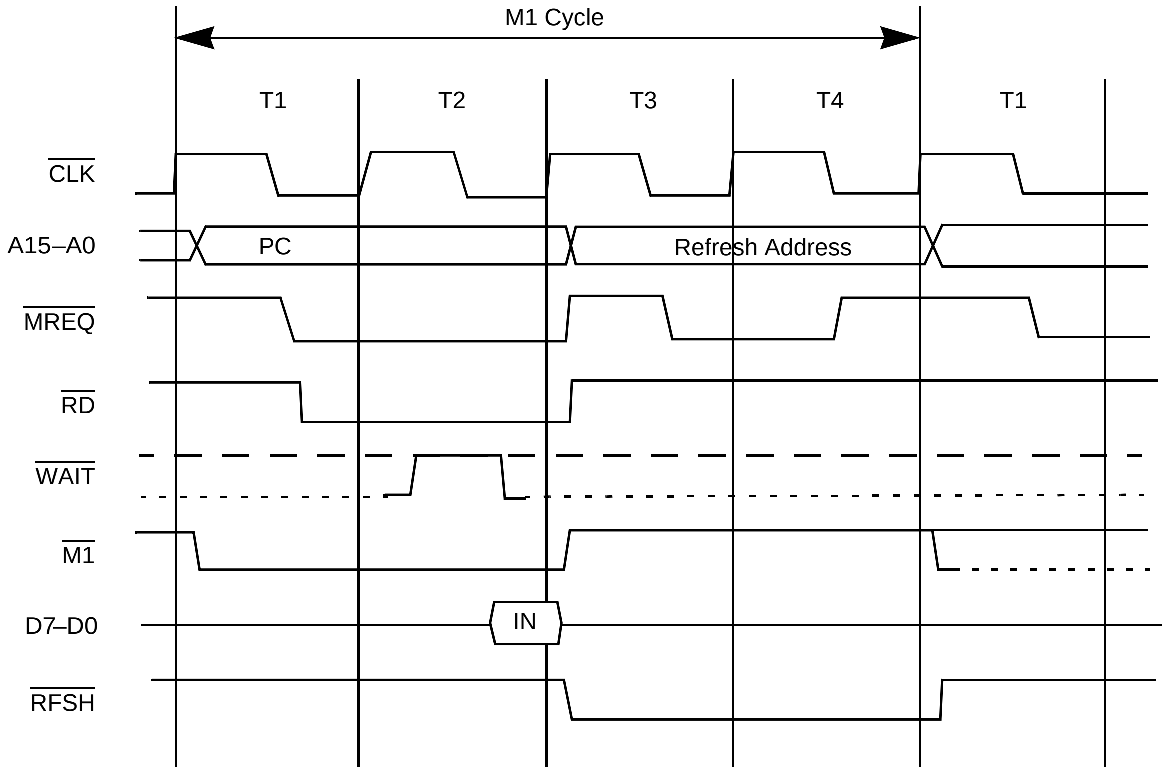


Figure 5. Instruction Op Code Fetch

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 $\overline{\text{WAIT}}$ signal. The $\overline{\text{MREQ}}$ signal and the $\overline{\text{RD}}$ signal are used the same way as in a fetch cycle. In a memory write cycle, the $\overline{\text{MREQ}}$ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The $\overline{\text{WR}}$ 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 $\overline{\text{WR}}$ 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.

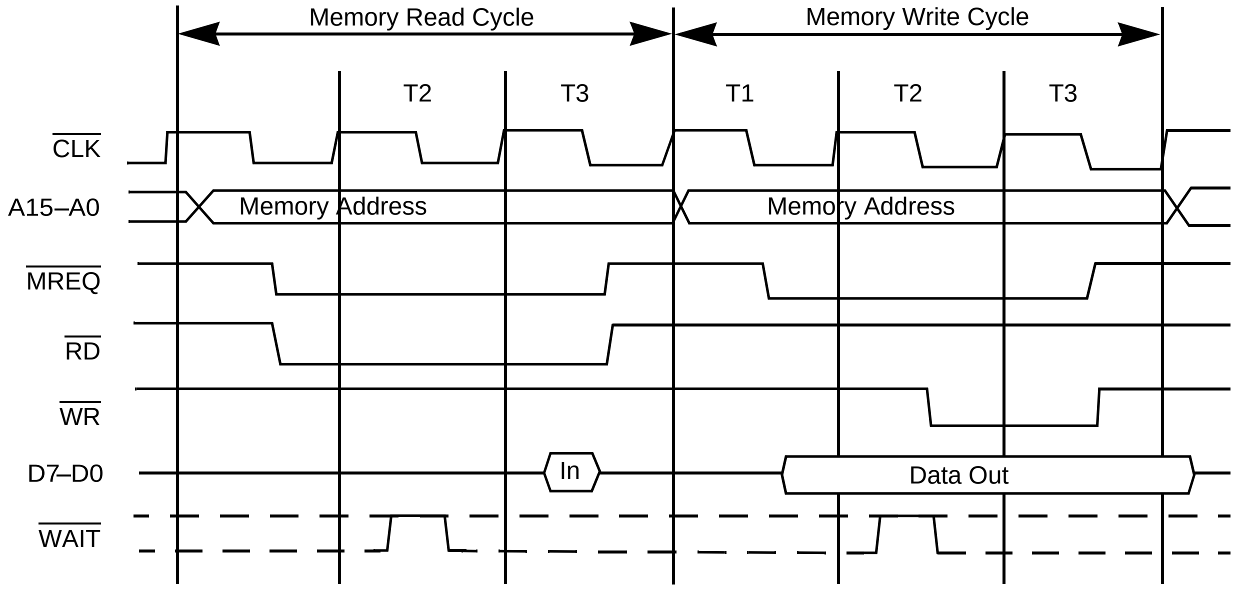


Figure 6. Memory Read or Write Cycle

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.

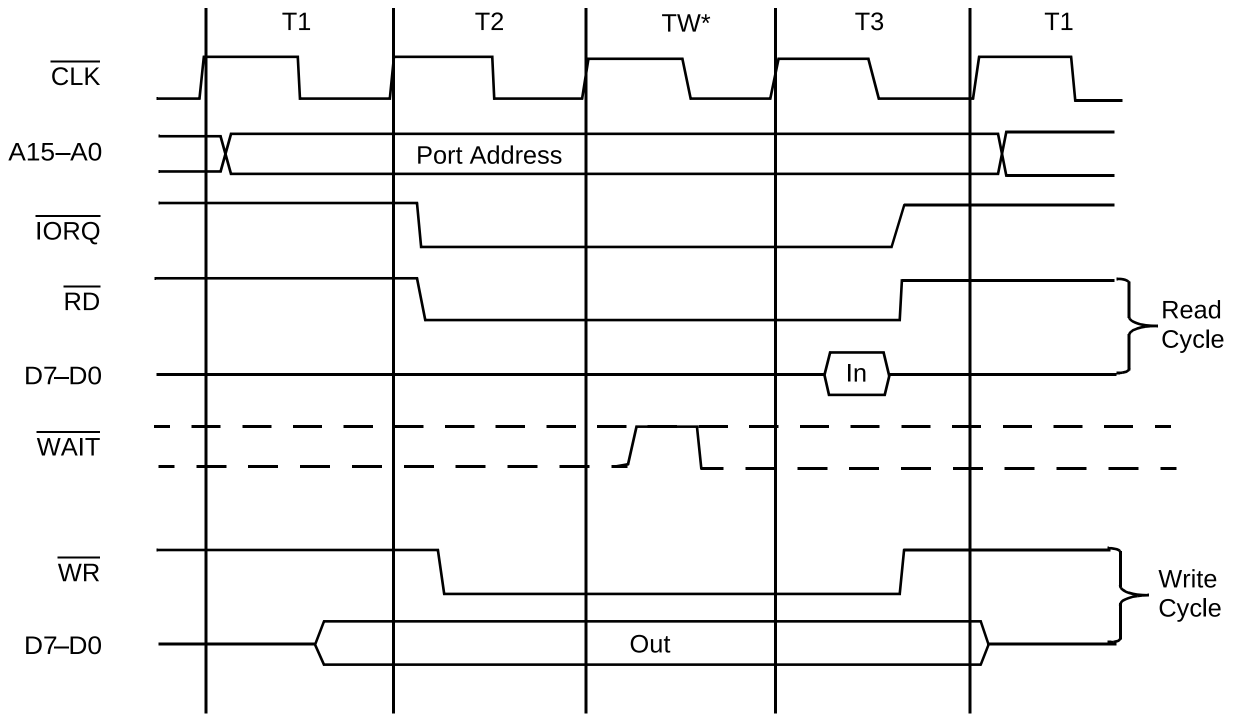


Figure 7. Input or Output Cycles

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 $\overline{\text{NMI}}$ or an $\overline{\text{INT}}$ signal.

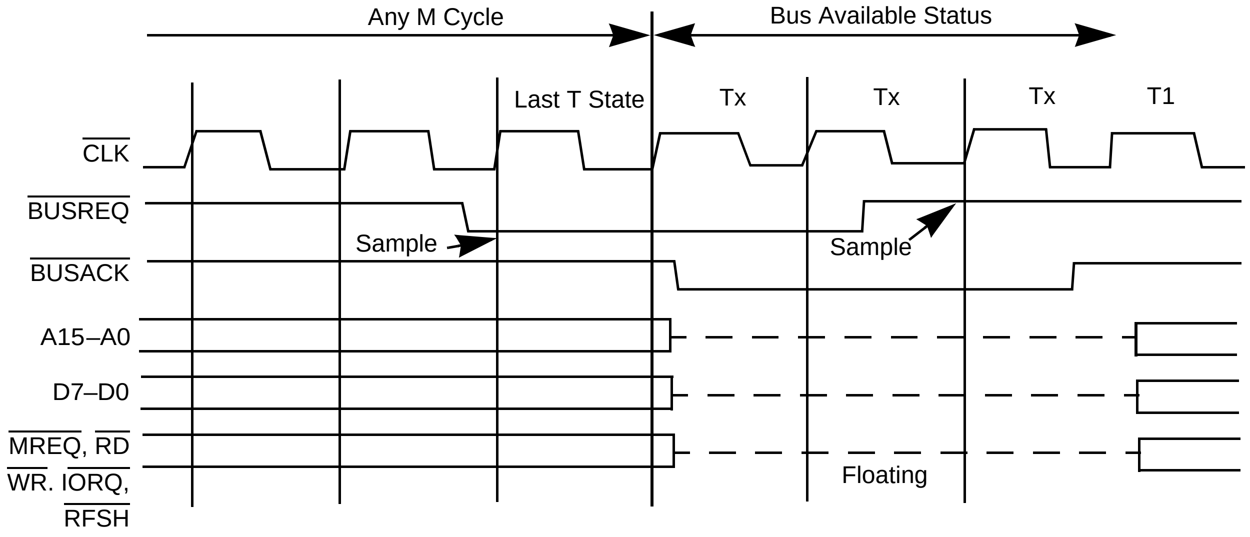


Figure 8. Bus Request/Acknowledge Cycle

Interrupt Request/Acknowledge Cycle

Figure 9 shows the timing associated with an interrupt cycle. The CPU samples the interrupt signal ($\overline{\text{INT}}$) 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 $\overline{\text{BUSREQ}}$ signal is active. When the signal is accepted, a special M1 cycle is generated. During this special M1 cycle, the $\overline{\text{IORQ}}$ signal becomes active (instead of the normal $\overline{\text{MREQ}}$) 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.

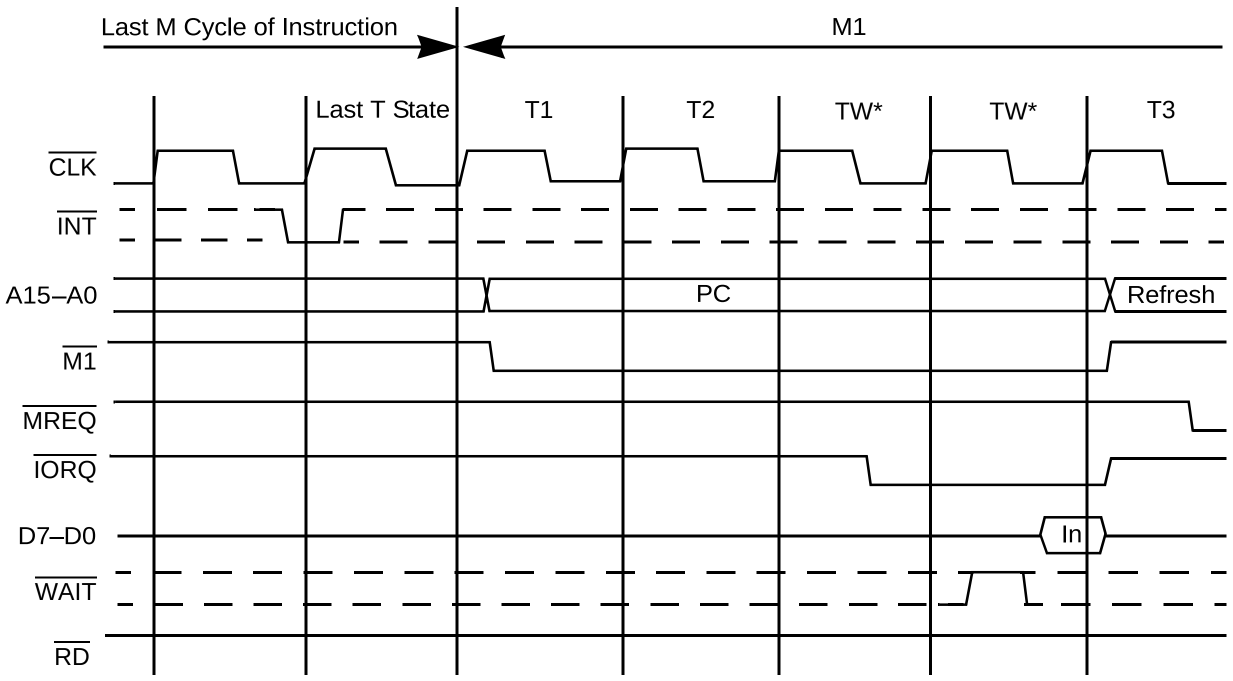


Figure 9. Interrupt Request/Acknowledge Cycle

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.

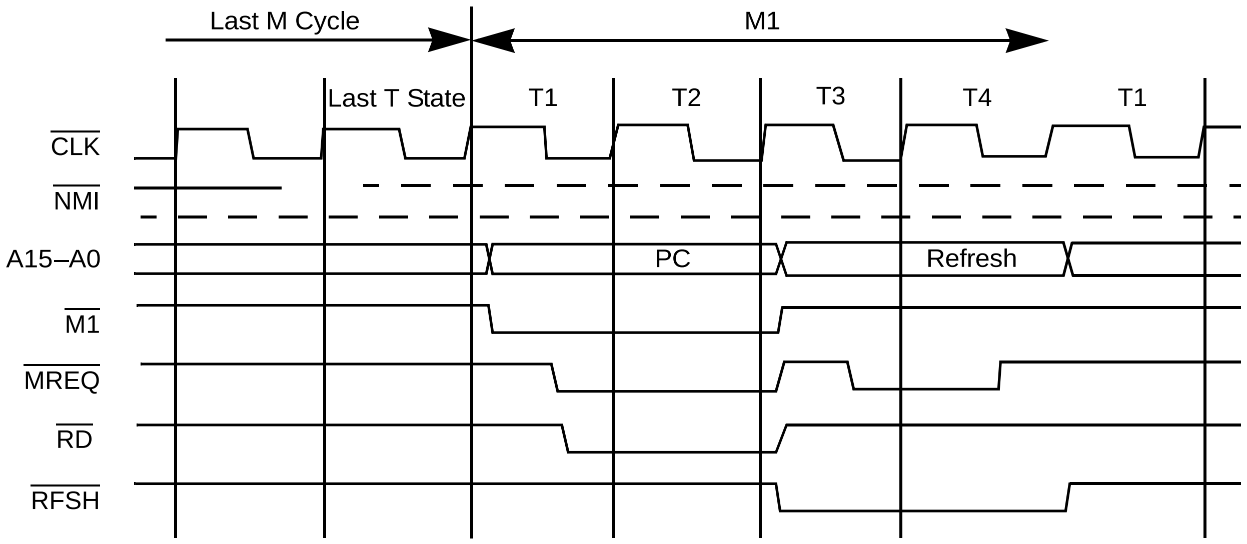


Figure 10. Nonmaskable Interrupt Request Operation

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.

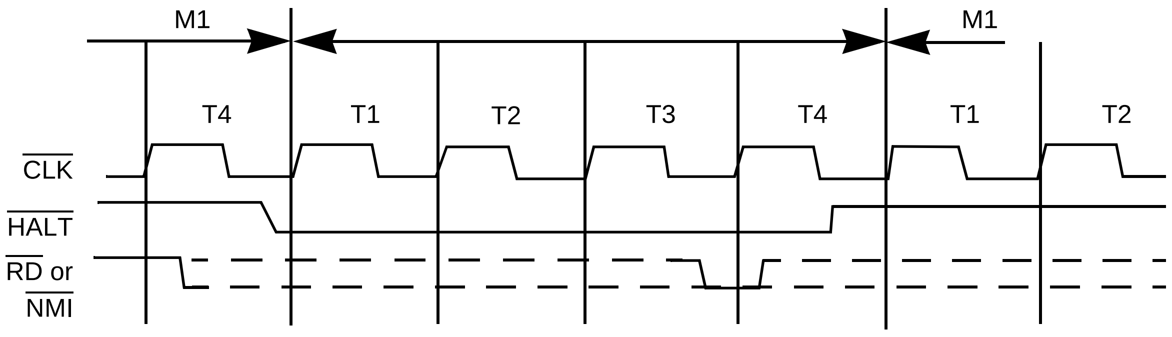


Figure 11. HALT Exit

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.

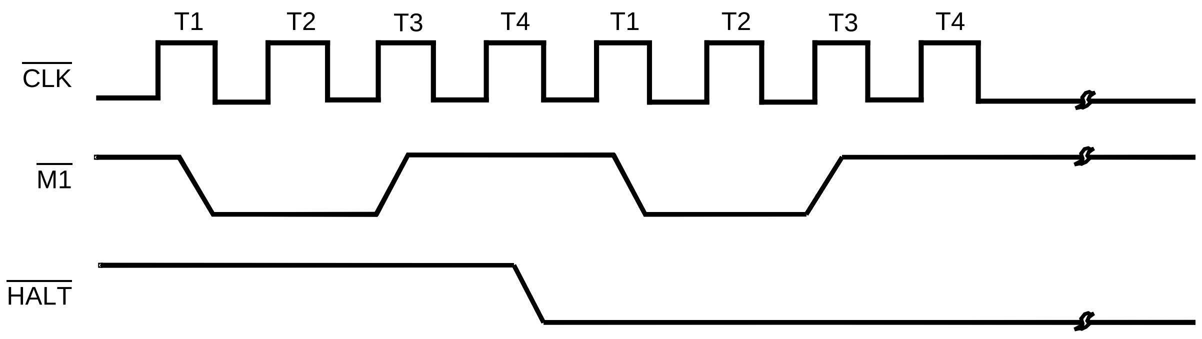


Figure 12. Power-Down Acknowledge

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.

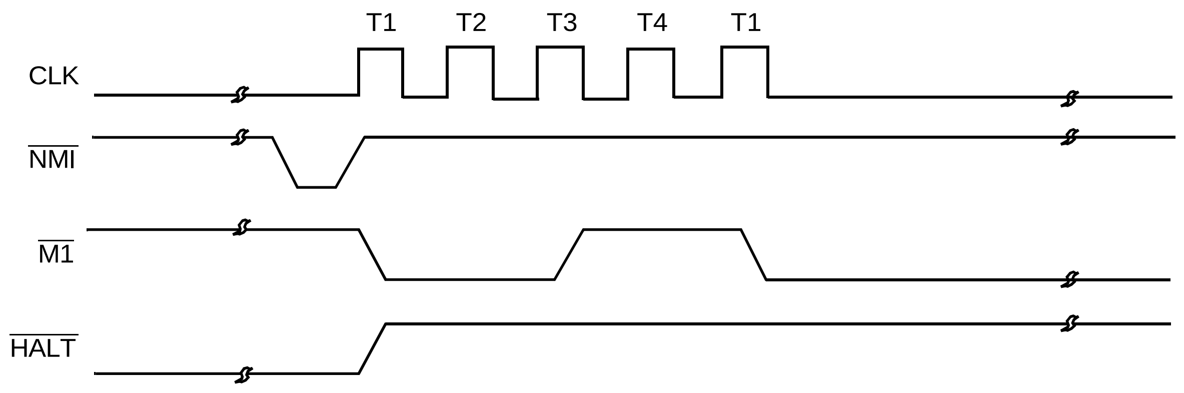


Figure 13. Power-Down Release Cycle, #1 of 3

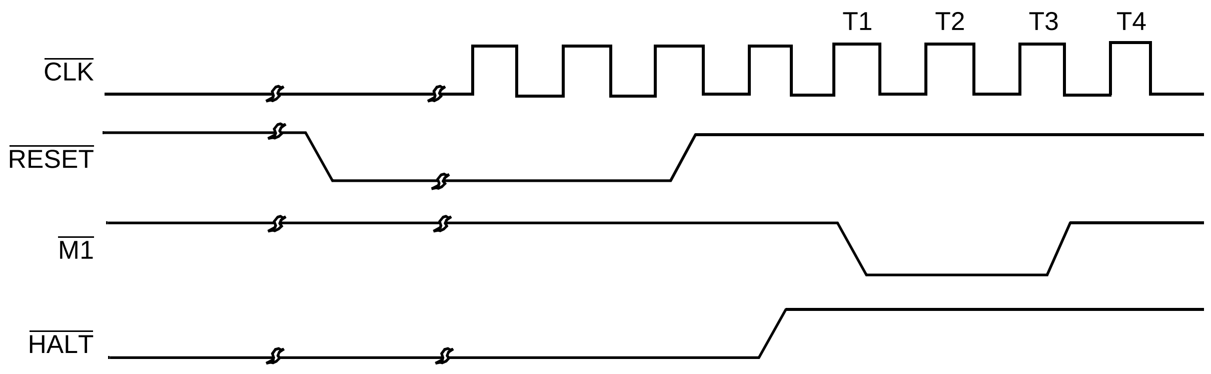


Figure 14. Power-Down Release Cycle, #2 of 3

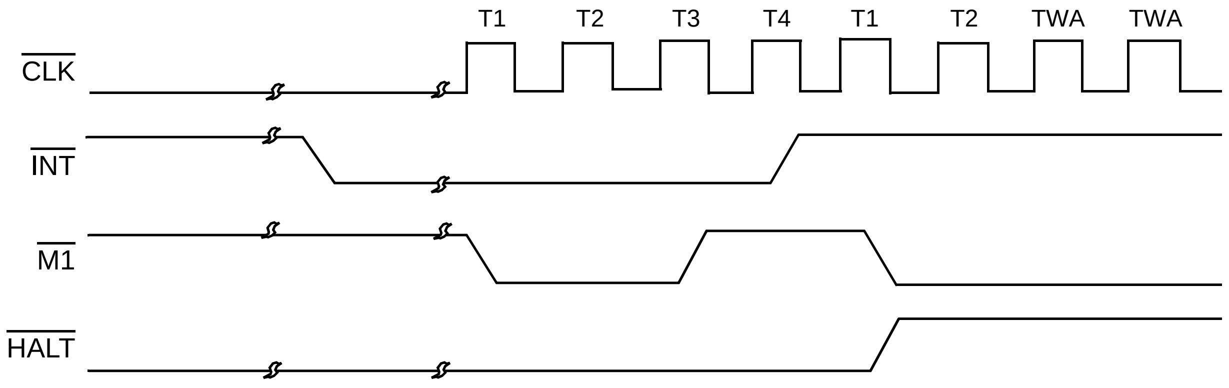


Figure 15. Power-Down Release Cycle, #3 of 3

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 ($\overline{\text{INT}}$) and a nonmaskable interrupt ($\overline{\text{NMI}}$). 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.

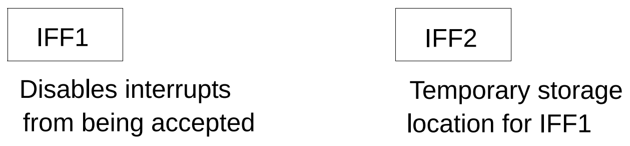

IFF1
Disables interrupts from being accepted
IFF2
Temporary storage location for IFF1

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, $\overline{\text{INT}}$ disabled.
DI Instruction Execution	0	0	Maskable $\overline{\text{INT}}$ disabled.
EI Instruction Execution	1	1	Maskable, $\overline{\text{INT}}$ 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.

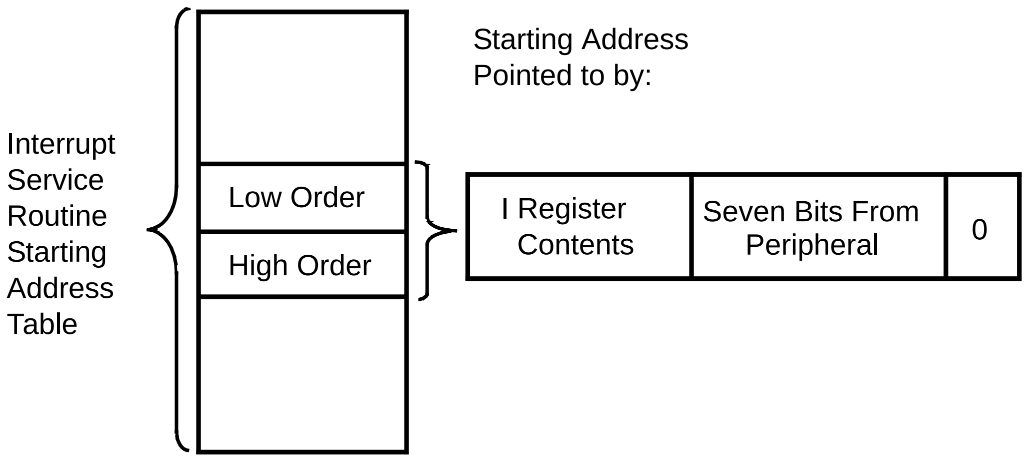


Figure 17. Mode 2 Interrupt Response Mode

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.

Hardware and Software Implementation

This chapter is an introduction to implementing systems that use the Z80 CPU. Figure 18 shows a simple Z80 system.

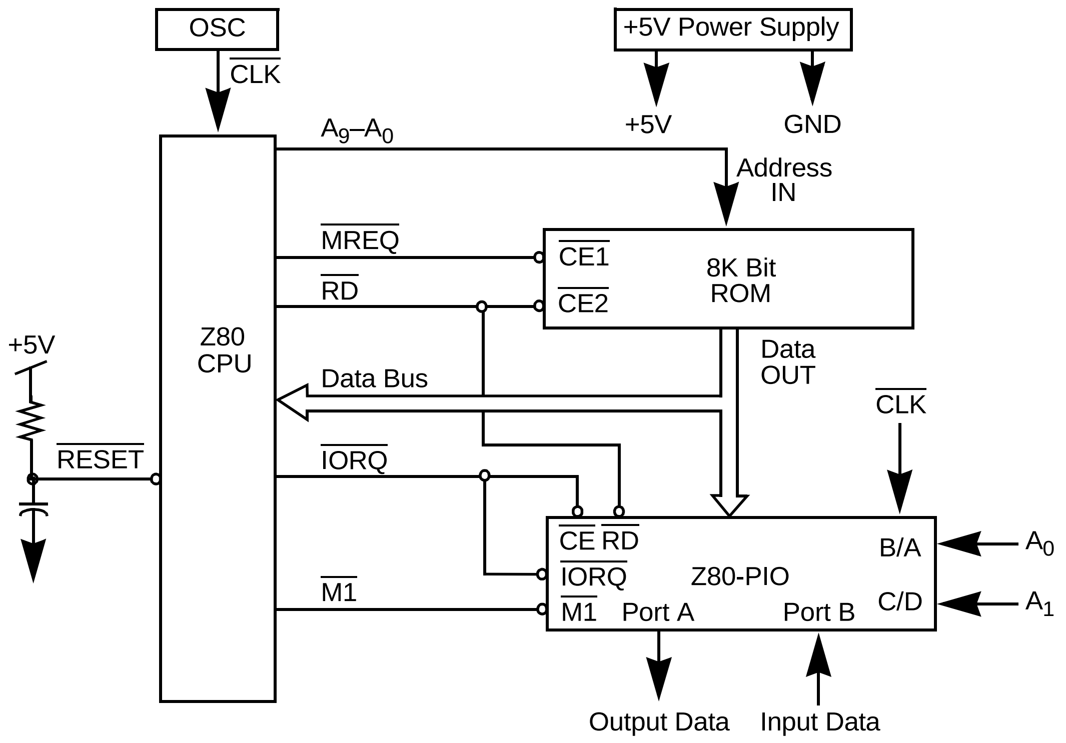


Figure 18. Minimum Z80 Computer System

Minimum System Hardware

Any Z80 system must include the following hardware elements:

- 5V power supply
- Oscillator
- Memory devices
- I/O circuits
- CPU

Because the Z80 CPU requires only a single 5V power supply, most small systems can be implemented using only this single supply.

The external memory can be any mixture of standard RAM, ROM, or PROM. In Figure 18, a single 8Kb (1KB) ROM comprises the entire memory system. The Z80 internal register configuration contains sufficient read/write storage, requiring no external RAM memory.

I/O circuits allow computer systems to interface with the external devices. In Figure 18, the output is an 8-bit control vector and the input is an 8-bit status word. The input data can be gated to the data bus using any standard three-state driver while the output data can be latched with any type of standard TTL latch. A Z80 PIO serves as the I/O circuit. This single circuit attaches to the data bus as indicated and provides the required 16 bits of TTL-compatible I/O. Refer to the Z80 CPU Peripherals User Manual (UM0081) to learn more about the operation of this circuit. This powerful computer is built with only three LSI circuits, a simple oscillator, and a single 5V power supply.

Adding RAM

Most computer systems require some external read/write memory for data storage and stack implementation. Figure 19 shows how 256 bytes of static memory are added to the example shown in Figure 18.

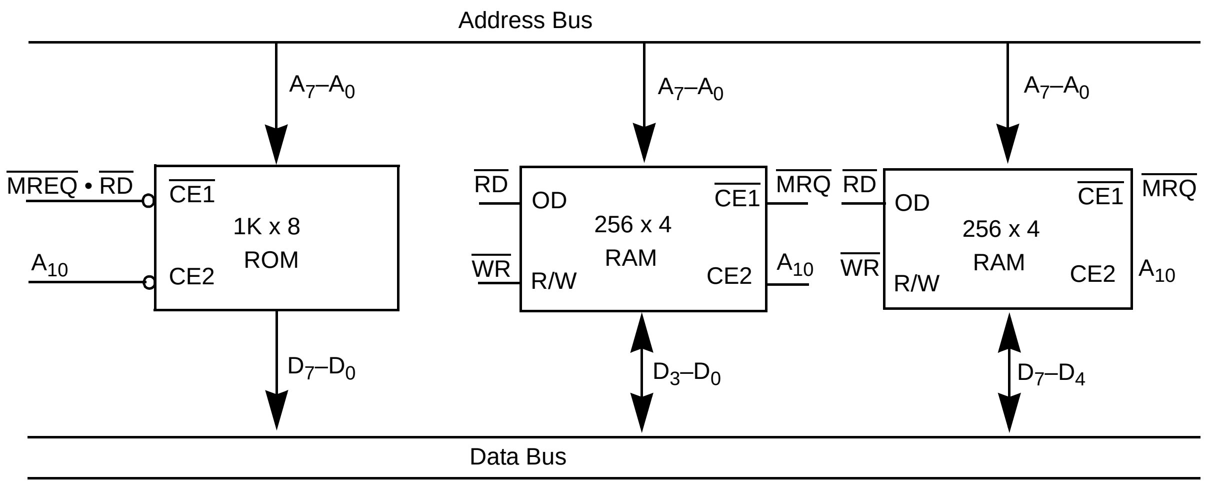


Figure 19. ROM and RAM Implementation

The memory space is assumed to be organized as shown in Figure 20.

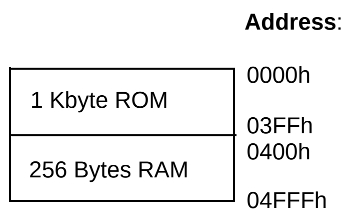


Figure 20. RAM Memory Space Organization

In Figure 20, the address space is portrayed in hexadecimal notation. Address bit A10 separates the ROM space from the RAM space, allowing this address to be used for the chip select function. For larger amounts of external ROM or RAM, a simple TTL decoder is required to form the chip selects.

Memory Speed Control

Slow memories can reduce costs for many applications. The $\overline{\text{WAIT}}$ line on the CPU allows the Z80 to operate with any speed memory. Memory access time requirements, which are covered in the Memory Read Or Write section on page 9, are most severe during the $\overline{\text{M1}}$ cycle instruction fetch. All other memory access cycles complete in an additional one half clock cycle. Hence, it is sometimes appropriate to add one wait state to the $\overline{\text{M1}}$ cycle so slower memories can be used.

Figure 21 is an example of a simple circuit that accomplishes this objective. This circuit can be changed to add a single wait state to any memory access, as indicated in Figure 22.

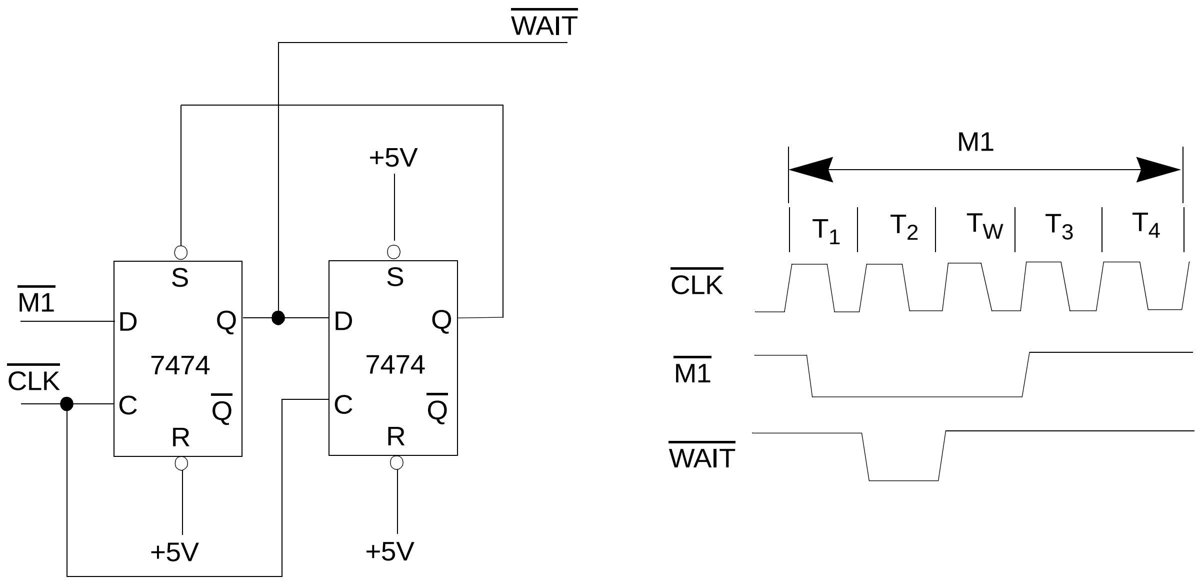


Figure 21. Adding One Wait State to an M1 Cycle

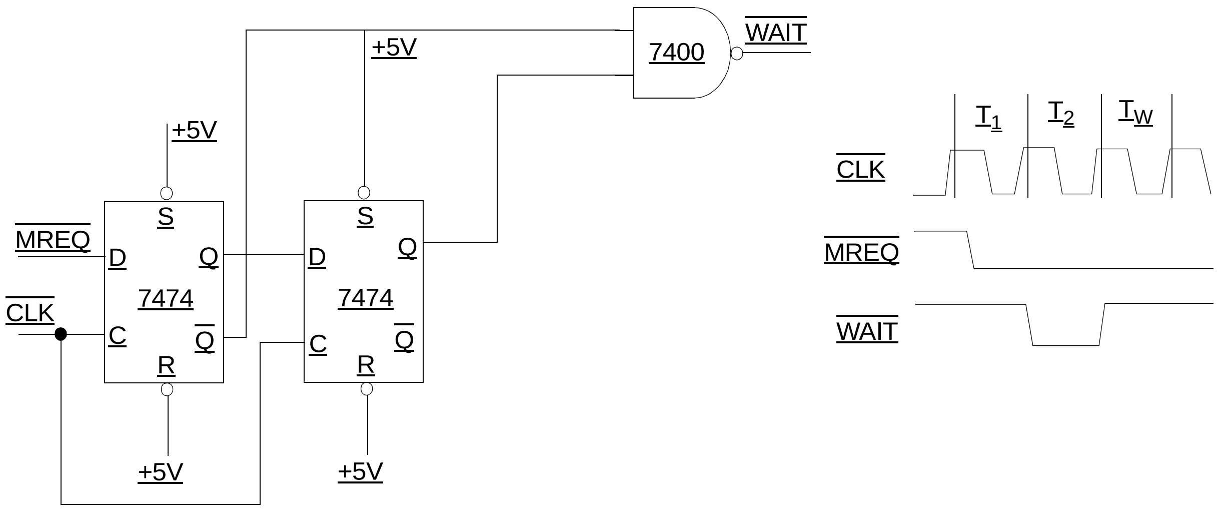


Figure 22. Adding One Wait State to Any Memory Cycle

Interfacing Dynamic Memories

Each individual dynamic RAM space includes its own specifications that require minor modifications to the examples provided here.

Figure 23 shows the logic necessary to interface 8KB of dynamic RAM using 18-pin 4K dynamic memories. This logic assumes that the RAMs are the only memory in the system so that A12 is used to select between the two pages of memory. During refresh time, all memories in the system must be read. The CPU provides the correct refresh address on lines A0 through A6. When adding more memory to the system, it is necessary to replace only the two gates that operate on A12 with a decoder that operates on all required address bits. Address buffers and data bus buffers are generally required for larger systems.

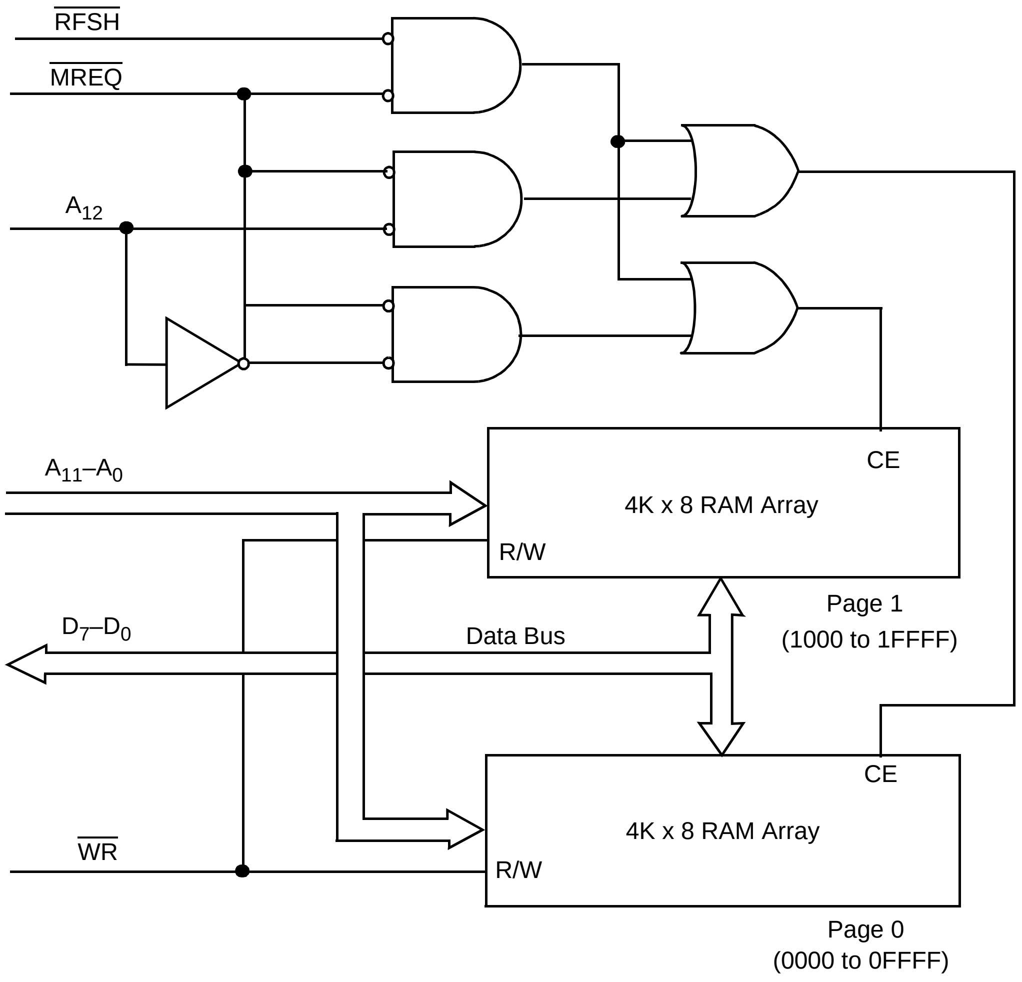


Figure 23. Interfacing Dynamic RAM Memory Spaces

Software Implementation Examples

The Z80 instruction set provides the user with a large number of operations to control the Z80 CPU. The main alternate and index registers can hold arithmetic and logical operations, form memory addresses, or act as fast-access storage for frequently used data.

Information can be moved directly from register to register, memory to memory, memory to registers, or from registers to memory. In addition, register contents and register/memory contents can be exchanged without using temporary storage. In particular, the contents of main and alternate registers can be completely exchanged by executing only two instructions, EX and EXX. This register exchange procedure can be used to separate the set of working registers from different logical procedures or to expand the set of available registers in a single procedure.

Storage and retrieval of data between pairs of registers and memory can be controlled on a last-in first-out basis through PUSH and POP instructions that utilize a special Stack Pointer (SP) Register. This stack register is available both to manipulate data and to automatically store and retrieve addresses for subroutine linkage. When a subroutine is called, for example, the address following the CALL instruction is placed on the top of the pushdown stack pointed to by SP. When a subroutine returns to the calling routine, the address on the top of the stack is used to set the program counter for the address of the next instruction. The stack pointer is adjusted automatically to reflect the current top stack position during PUSH, POP, CALL, and RET instructions. This stack mechanism allows pushdown data stacks and subroutine calls to be nested to any practical depth because the stack area can potentially be as large as memory space.

The sequence of instruction execution can be controlled by six different flags (carry, zero, sign, parity/overflow, add/subtract, half-carry), which reflect the results of arithmetic, logical, shift, and compare instructions. After the execution of an instruction that sets a flag, that flag can be used to control a conditional jump or return instruction. These instructions provide logical control following the manipulation of single bit, 8-bit byte, or 18-bit data quantities.

A full set of logical operations, including AND, OR, XOR (exclusive-OR), CPL (NOR), and NEG (two’s complement) are available for Boolean operations between the Accumulator and all other 8-bit registers, memory locations, or immediate operands.

In addition, a full set of arithmetic and logical shifts in both directions are available which operate on the contents of all 8-bit primary registers or directly on any memory location. The carry flag can be included or set by these shift instructions to provide both the testing of shift results and to link register/register or register/memory shift operations.

Specific Z80 Instruction Examples

Example 1

When a 737-byte data string in memory location DATA must be moved to location BUFFER, the operation is programmed as follows:

```
  LD      HL, DATA      ;START ADDRESS OF DATA STRING
  LD      DE, BUFFER    ;START ADDRESS OF TARGET BUFFER
  LD      BC, 737       ;LENGTH OF DATA STRING
  LDIR                  ;MOVE STRING–TRANSFER MEMORY POINTED
                        ;TO BY HL INTO MEMORY LOCATION POINTED
                        ;TO BY DE INCREMENT HL AND DE,
                        ;DECREMENT BC PROCESS UNTIL BC = 0
```

Eleven bytes are required for this operation and each byte of data is moved in 21 clock cycles.

Example 2

A string in memory (limited to a maximum length of 132 characters) starting at location DATA is to be moved to another memory location starting at location BUFFER until an ASCII $ (used as a string delimiter) is found. This operation is performed as follows:

```
  LD      HL, DATA      ;STARTING ADDRESS OF DATA STRING
  LD      DE, BUFFER    ;STARTING ADDRESS OF TARGET BUFFER
  LD      BC, 132       ;MAXIMUM STRING LENGTH
  LD      A, '$'        ;STRING DELIMITER CODE
LOOP:     CP (HL)       ;COMPARE MEMORY CONTENTS WITH
                        ;DELIMITER
  JR      Z, END-$      ;GO TO END IF CHARACTERS EQUAL
  LDI                   ;MOVE CHARACTER (HL) to (DE)
                        ;INCREMENT HL AND DE, DECREMENT BC
  JP      PE, LOOP      ;GO TO LOOP IF MORE CHARACTERS
END:                    ;OTHERWISE, FALL THROUGH
                        ;NOTE: P/V FLAG IS USED
                        ;TO INDICATE THAT REGISTER BC WAS
                        ;DECREMENTED TO ZERO
```

Nineteen bytes are required for this operation.

Example 3

A 16-digit decimal number is shifted as depicted in Figure 24. This shift is performed to mechanize BCD multiplication or division. The 16-digit decimal number is represented in packed BCD format (two BCD digits/byte) The operation is programmed as follows:

```
  LD      HL, DATA      ;ADDRESS OF FIRST BYTE
  LD      B, COUNT      ;SHIFT COUNT
  XOR     A             ;CLEAR ACCUMULATOR
ROTAT:    RLD           ;ROTATE LEFT low-order DIGIT IN ACC
                        ;WITH DIGITS IN (HL)
  INC     HL            ;ADVANCE MEMORY POINTER.
  DJNZ    ROTAT-$       ;DECREMENT B AND GO TO ROTAT IF
                        ;B IS NOT ZERO, OTHERWISE FALL
                        ;THROUGH
```

Eleven bytes are required for this operation.

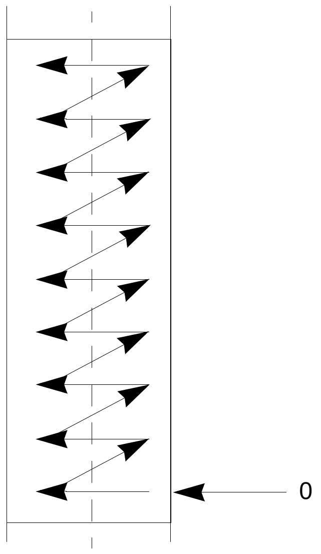


Figure 24. Shifting of BCD Digits/Bytes

Example 4

One number is to be subtracted from another number, both of which exist in packed BCD format and are of equal but varying length. The result is stored in the location of the minuend. The operation is programmed as follows:

```
  LD      HL, ARG1      ;ADDRESS OF MINUEND
  LD      DE, ARG2      ;ADDRESS OF SUBTRAHEND
  LD      B, LENGTH     ;LENGTH OF TWO ARGUMENTS
  AND     A             ;CLEAR CARRY FLAG
SUBDEC:LD A, (DE)       ;SUBTRAHEND TO ACC
```

```
SBC     A, (HL)      ;SUBTRACT (HL) FROM ACC
DAA                  ;ADJUST RESULT TO DECIMAL CODED VALUE
LD      (HL), A      ;STORE RESULT
INC     HL           ;ADVANCE MEMORY POINTERS
INC     DE
DJNZ    SUBDEC–$     ;DECREMENT B AND GO TO SUBDEC
                     ;IF B
                     ;NOT ZERO, OTHERWISE FALL
                     ;THROUGH
```

Seventeen bytes are required for this operation.

Programming Task Examples

As indicated in Table 2, this example program sorts an array of numbers to ascending order, using a standard exchange sorting algorithm. These numbers range from 0 to 255.

Table 2. Bubble Listing

Location	Object Code	Statement	Source Statement		
		1	;	standard exchange (bubble) sort routine	
		2	;		
		3	;	at entry:	hl contains address of data
		4			c contains number of elements to be sorted
		5			(1 < c < 256)
		6	;		
		7	;	at exit	data sorted in ascending order
		8	;		
		9	;	use of registers	
		10	;		
		11	;	register	contents
		12	;		
		13	;	a	temporary storage for calculations
		14	;	b	counter for data array
		15	;	c	length of data array
		16	;	d	first element in comparison
		17	;	e	second element in comparison
		18	;	h	flag to indicate exchange

Table 2. Bubble Listing (Continued)

Location	Object Code	Statement	Source Statement			
		19	;	l	unused	
		20	;	ix	pointer into data array	
		21	;	iy	unused	
		22	;			
0000	222600	23	sort:	ld	(data), hl	; save data address
0003	cb84	24	loop:	res	flag, h	; initialize exchange flag
0005	41	25		ld	b, c	; initialize length counter
0006	05	26		dec	b	; adjust for testing
0007	dd2a260 0	27		ld	ix, (data)	; initialize array pointer
000b	dd7e00	28	next:	ld	a, (ix)	; first element in comparison
000e	57	29		ld	d, a	; temporary storage for element
goof	dd5e01	30		ld	e, (ix+1)	; second element in comparison
0012	93	31		sub	e	; comparison first to second
0013	3008	32		jr	pc, noex–$	; if first > second, no jump
0015	dd7300	33		ld	(ix), e	; exchange array elements
0018	dd7201	34		ld	(ix+i), d	
001b	cbc4	35		set	flag, h	; record exchange occurred
0010	dd23	36	noex:	inc	ix	; point to next data element
001f	10ea	37		djnz	next–$	; count number of comparisons
		38				; repeat if more data pairs
0021	cb44	39		bit	flag, h	; determine if exchange occurred
0023	20de	40		jr	nz, loop–$	; continue if data unsorted
0025	c9	41		ret		; otherwise, exit
		42	;			
0026		43	flag:	equ	0	; designation of flag bit
0026		44	data:	defs	2	; storage for data address
		45		end		

The program outlined in Table 3 multiplies two unsigned 16-bit integers, leaving the result in the HL register pair.

Table 3. Multiply Listing

Location	Object Code	Statement	Source Statement			
0000		1	mult:;	unsigned sixteen bit integer multiply.		
		2	;	on entrance: multiplier in de.		
		3	;	multiplicand in hl.		
		4	;			
		5	;	on exit	result in hl.	
		6	;			
		7	;	register	uses:	
		8	;			
		9	;			
		10	;	h	high-order partial result	
		11	;	l	low-order partial result	
		12	;	d	high-order multiplicand	
		13	;	e	low-order multiplicand	
		14	;	b	counter for number of shifts	
		15	;	c	high-order bits of multiplier	
		16	;	a	low-order bits of multiplier	
		17	;			
0000	0610	18		ld	b, 16;	number of bits-initialize
0002	4a	19		ld	c, d;	move multiplier
0003	7b	20		ld	a, e;	
0004	eb	21		ex	de, hl;	move multiplicand
0005	210000	22		ld	hl, 0;	clear partial result
0008	cb39	23	mloop:	srl	c;	shift multiplier right
000a	if	24		rra		least-significant bit is
		25	;			in carry.
000b	3001	26		jr	nc, noadd–$;	if no carry, skip the add.
good	19	27		add	hl, de;	else add multiplicand to
		28	;			partial result.
000e	eb	29	noadd:	ex	de, h l;	shift multiplicand left
goof	29	30		add	hl, hl;	by multiplying it by two.
0010	eb	31		ex	de, hl;	
0011	10f5	32		djnz	mloop–$;	repeat until no more bits.
0013	c9	33		ret;		
		34		end;		

Z80 CPU Instructions

The Z80 CPU can execute 158 different instruction types including all 78 of the 8080A CPU. The instructions fall into these major groups:

- Load and Exchange
- Block Transfer and Search
- Arithmetic and Logical
- Rotate and Shift
- Bit Manipulation (Set, Reset, Test)
- Jump, Call, and Return
- Input/Output
- Basic CPU Control

Instruction Types

The load instructions move data internally among CPU registers or between CPU registers and external memory. All of these instructions specify a source location from which the data is to be moved, and a destination location. The source location is not altered by a load instruction. Examples of load group instructions include moves between any of the general-purpose registers such as move the data to Register B from Register C. This group also includes load-immediate to any CPU register or to any external memory location. Other types of load instructions allow transfer between CPU registers and memory locations. The exchange instructions can trade the contents of two registers.

A unique set of block transfer instructions is provided in the Z80 CPU. With a single instruction, a block of memory of any size can be moved to any other location in memory. This set of block moves is extremely valuable when processing large strings of data. With a single instruction, a block of external memory of any required length can be searched for any 8-bit character. When the character is found or the end of the block is reached, the instruction automatically terminates. Both the block transfer and the block search instructions can be interrupted during their execution so they are not occupying the CPU for long periods of time.

The arithmetic and logical instructions operate on data stored in the Accumulator and other general-purpose CPU registers or external memory locations. The results of the operations are placed in the Accumulator and the appropriate flags are set according to the result of the operation.

An example of an arithmetic operation is adding the Accumulator to the contents of an external memory location. The results of the addition are placed in the Accumulator. This group also includes 16-bit addition and subtraction between 16-bit CPU registers.

The rotate and shift group allows any register or any memory location to be rotated right or left, with or without carry, and either arithmetic or logical. Additionally, a digit in the Accumulator can be rotated right or left with two digits in any memory location.

The bit manipulation instructions allow any bit in the Accumulator, any general-purpose register, or any external memory location to be set, reset, or tested with a single instruction. For example, the most-significant bit of Register H can be reset. This group is especially useful in control applications and for controlling software flags in general-purpose programming.

The JUMP, CALL, and RETURN instructions are used to transfer between multiple locations in the user's program. This group uses several different techniques for obtaining the new program counter address from specific external memory locations. A unique type of call is the RESTART instruction. This instruction actually contains the new address as a part of the 8-bit op code. This instruction is possible because only eight separate addresses located in Page 0 of external memory can be specified. Program jumps can also be achieved by loading Register HL, IX, or IY directly into the Program Counter, which allows the jump address to be a complex function of the routine being executed.

The input/output group of instructions in the Z80 CPU allow for a wide range of transfers between external memory locations or the general-purpose CPU registers, and the external I/O devices. In each case, the port number is provided on the lower eight bits of the address bus during any I/O transaction. One instruction allows this port number to be specified by the second byte of the instruction while other Z80 instructions allow it to be specified as the contents of the C Register. One major advantage of using the C register as a pointer to the I/O device is that it allows multiple I/O ports to share common software driver routines. This advantage is not possible when the address is part of the op code if the routines are stored in ROM. Another feature of these input instructions is the automatic setting of the Flag Register, making additional operations unnecessary to determine the state of the input data. The parity state is one example.

The Z80 CPU includes single instructions that can move blocks of data (up to 256 bytes) automatically to or from any I/O port directly to any memory location. In conjunction with the dual set of general-purpose registers, these instructions provide fast I/O block transfer rates. The power of this I/O instruction set is demonstrated by the Z80 CPU providing all required floppy disk formatting on double-density floppy disk drives on an interrupt-driven basis. For example, the CPU provides the preamble, address, data, and enables the CRC codes.

Finally, the basic CPU control instructions allow multiple options and modes. This group includes instructions such as setting or resetting the interrupt enable flip-flop or setting the mode of interrupt response.

Addressing Modes

Most of the Z80 instructions operate on data stored in internal CPU registers, external memory, or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section is a brief summary of the types of addressing used in the Z80 CPU while subsequent sections detail the type of addressing available for each instruction group.

Immediate Addressing

In the Immediate Addressing Mode, the byte following the op code in memory contains the actual operand, as shown in Figure 25.

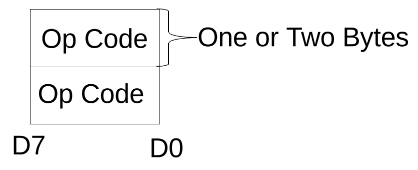


Figure 25. Immediate Addressing Mode

An example of this type of instruction is to load the Accumulator with a constant, in which the constant is the byte immediately following the op code.

Immediate Extended Addressing

This mode is an extension of immediate addressing in that the two bytes following the op codes are the operand, as shown in Figure 26.

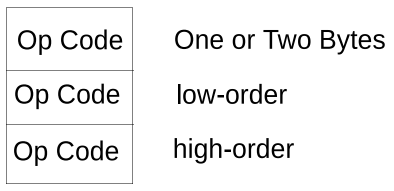


Figure 26. Immediate Extended Addressing Mode

An example of this type of instruction is to load the HL register pair (16-bit register) with 16 bits (two bytes) of data.

Modified Page Zero Addressing

The Z80 contains a special single-byte CALL instruction to any of eight locations in Page 0 of memory. This instruction, which is referred to as a restart, sets the Program Counter to an effective address in Page 0. The value of this instruction is that it allows a single byte to specify a complete 16-bit address at which commonly-called subroutines are located, thereby saving memory space.

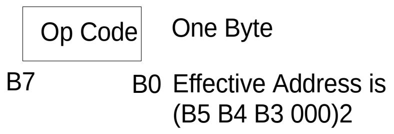


Figure 27. Modified Page Zero Addressing Mode

Relative Addressing

Relative addressing uses one byte of data following the op code to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two’s complement number that is added to the address of the op code of the following instruction.

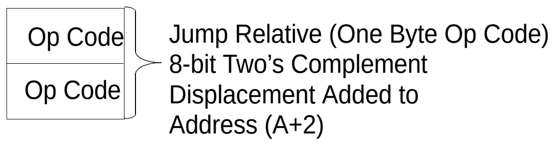


Figure 28. Relative Addressing Mode

The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most programs, relative jumps are by far the most prevalent type of jump due to the proximity of related program segments. Therefore, these instructions can significantly reduce memory space requirements. The signed displacement can range between +127 and –128 from A+2. This range allows for a total displacement of +129 to –126 from the jump relative op code address. Another major advantage is that it allows for relocatable code.

Extended Addressing

Extended Addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address at which an operand is located.

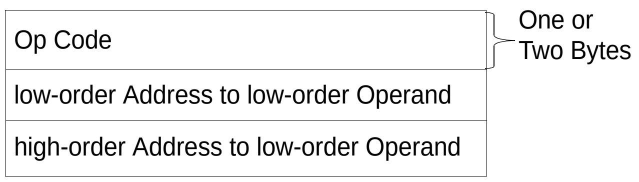

Op Code
One or
Two Bytes
low-order Address to low-order Operand
high-order Address to low-order Operand

Figure 29. Extended Addressing Mode

Extended addressing is required for a program to jump from any location in memory to any other location, or load and store data in any memory location.

During extended addressing use, specify the source or destination address of an operand. This notation (*nn*) is used to indicate the contents of memory at *nn*, in which *nn* is the 16-bit address specified in the instruction. The two bytes of address *nn* are used as a pointer to a memory location. The parentheses always indicates that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

Indexed Addressing

In the Indexed Addressing Mode, the byte of data following the op code contains a displacement that is added to one of the two index registers (the op code specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation.

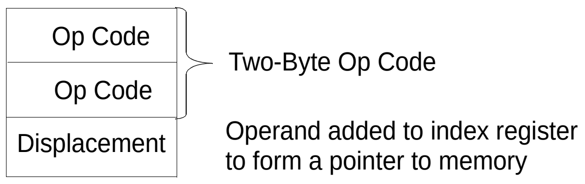


Figure 30. Indexed Addressing Mode

An example of an indexed instruction is to load the contents of the memory location (Index Register + Displacement) into the Accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data because the index register can point to the start of any table. Two index registers are provided because often operations require two or more tables. Indexed addressing also allows for relocatable code.

The two index registers in the Z80 CPU are referred to as IX and IY. To indicate indexed addressing, use the following notation:

(IX+d) or *(IY+d)*

In this notation, *d* is the displacement specified after the op code. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing

Many of the Z80 op codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing is to load the data in Register 6 into Register C.

Implied Addressing

Implied addressing refers to operations in which the op code automatically implies one or more CPU registers as containing the operands. An example is the set of arithmetic operations in which the Accumulator is always implied to be the destination of the results.

Register Indirect Addressing

This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is powerful and it is used in a wide range of applications.

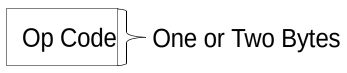


Figure 31. Register Indirect Addressing Mode

An example of this type of instruction is to load the Accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of Register Indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for powerful but simple to implement memory accesses. The block move and search commands in the Z80 CPU are extensions of this type of addressing in which automatic register incrementing, decrementing, and comparing is added. The notation for indicating Register Indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol (HL) specifies that the contents of the HL register are to be used as a pointer to a memory location. Often Register Indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the lower order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

Bit Addressing

The Z80 contains a large number of bit set, reset, and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, Register Indirect, and indexed) while three bits in the op code specify which of the eight bits is to be manipulated.

Addressing Mode Combinations

Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing can be employed. For example, load can use immediate addressing to specify the source and Register Indirect or indexed addressing to specify the destination.

Instruction Op Codes

This section describes each of the Z80 instructions and provides tables listing the op codes for every instruction. In each of these tables, the op codes in shaded areas are identical to those offered in the 8080A CPU. Also depicted is the assembly language mnemonic that is used for each instruction. All instruction op codes are listed in hexadecimal notation. Single-byte op codes require two hex characters while double byte op codes require four hex characters. For convenience, the conversion from hex to binary is repeated in Table 4.

Table 4. Hex, Binary, Decimal Conversion Table

Hex		Binary		Decimal
0	=	0000	=	0
1	=	0001	=	1
2	=	0010	=	2
3	=	0011	=	3
4	=	0100	=	4
5	=	0101	=	5
6	=	0110	=	6
7	=	0111	=	7

Table 4. Hex, Binary, Decimal Conversion Table (Continued)

Hex		Binary		Decimal
8	=	1000	=	8
9	=	1001	=	9
A	=	1010	=	10
B	=	1011	=	11
C	=	1100	=	12
D	=	1101	=	13
E	=	1110	=	14
F	=	1111	=	15

The Z80 instruction mnemonics consist of an op code and zero, one, or two operands. Instructions in which the operand is implied contains no operand. Instructions that contain only one logical operand, in which one operand is invariant (such as the Logical OR instruction), are represented by a one-operand mnemonic. Instructions that contain two varying operands are represented by two operand mnemonics.

Load and Exchange

Table 5 defines the op codes for all of the 8-bit load instructions implemented in the Z80 CPU. Also described in this table is the type of addressing used for each instruction. The source of the data is found on the top horizontal row and the destination is specified in the left column. For example, load Register C from Register B uses the op code `48h`. In all of the figures, the op code is specified in hexadecimal notation and the `48h` (`0100 1000` binary) code is fetched by the CPU from external memory during M1 time, decoded, and then the register transfer is automatically performed by the CPU.

The assembly language mnemonic for this entire group is LD, followed by the destination, followed by the source (LD DEST, SOURCE).

Note: Several combinations of addressing modes are possible. For example, the source can use register addressing and the destination can be registered indirect; such as load the memory location pointed to by Register HL with the contents of the D Register. The op code for this operation is 72. The mnemonic for this load instruction is LD *(HL), D*.

Table 5. 8-Bit Load Group LD

		Source															
		Implied		**Register**							**Reg Indirect**			**Indexed**		**Ext.**	**Imm.**
Destination		**I**	**R**	**A**	**B**	**C**	**D**	**E**	**F**	**L**	**(HL)**	**(BC)**	**(DE)**	**(IX+d)**	**(IY+d)**	**Inn)**	**n**
Register	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	FD 7E d	DD 7E d	FD 3A nn	FD 2E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		DD D5 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		DD DE n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		DD 1B n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		DD 1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		DD 2B n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		DD 36 n
Register Indirect	(HL)			77	70	71	72	73	74	75							DD 78 D
	(BC)			02													
	(DE)			12													
Indexed	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
Ext. Addr.	(nn)			32 n n				.									
Implied	I			ED 47													
	R			ED 4F													

Note: Descriptions of the 8-Bit Load Group instructions begin on page 68.

The parentheses around the HL indicate that the contents of HL are used as a pointer to a memory location. In all Z80 load instruction mnemonics, the destination is always listed first, with the source following. The Z80 assembly language is defined for ease of programming. Every instruction is self documenting and programs written in Z80 language are easy to maintain.

In Table 5, some op codes that are available in the Z80 CPU use two bytes. This feature is an efficient method of memory utilization because 8-, 18-, 24-, or 32-bit instructions are implemented in the Z80 CPU. Often utilized instructions such as arithmetic or logical operations are only eight bits, which results in better memory utilization than is achieved with fixed instruction sizes such as 16 bits.

All load instructions using indexed addressing for either the source or destination location actually use three bytes of memory, with the third byte being the displacement, *d*. For example, a Load Register E instruction with the operand pointed to by IX with an offset of +8 is written as:

LID *E, (IX + 8)*

The instruction sequence for this value in memory is shown in Figure 32.

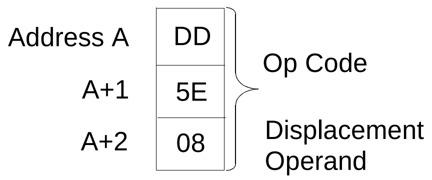


Figure 32. Example of a 3-Byte Load Indexed Instruction Sequence

The two extended addressing instructions are also three-byte instructions. For example, the instruction to load the Accumulator with the operand in memory location `6F32h` is written as:

```
LID A, (6F 32h)
```

The instruction sequence for this value in memory is shown in Figure 33.

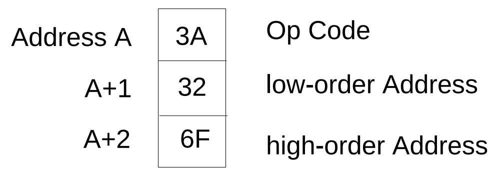


Figure 33. Example of a 3-Byte Load Extended Instruction Sequence

In this figure, note that the low-order portion of the address is always the first operand.

The load immediate instructions for the general-purpose 8-bit registers are two-byte instructions. The instruction for loading Register H with the value `36h` is written as:

```
LD H, 36h
```

The instruction sequence for this value in memory is shown in Figure 34.

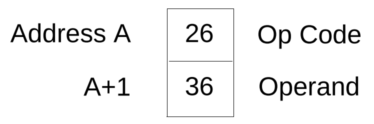


Figure 34. Example of a 2-Byte Load Immediate Instruction Sequence

Loading a memory location using indexed addressing for the destination and immediate addressing for the source requires four bytes. For example:

```
LD (IX–15), 21h
```

The instruction sequence for this value in memory is shown in Figure 35.

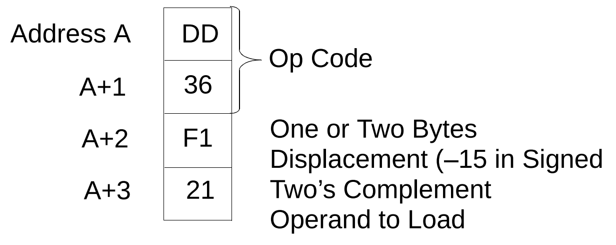


Figure 35. Example of a 4-Byte Load Indexed/Immediate Instruction Sequence

In this figure, note that with any indexed addressing, the displacement always follows directly after the op code.

Table 6 specifies the 16-bit load operations, for which the extended addressing feature covers all register pairs. Register indirect operations specifying the stack pointer are the PUSH and POP instructions. The mnemonic for these instructions is PUSH and POP.

Table 6. 16-Bit Load Group LD, PUSH, and POP

			Source								
			Register						Imm. Ext.	Ext.	Reg. Indir.
	Register		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
		AF										P1
		BC								01 n n	ED 4B n n	C1
		DE								11 n n	ED 5B n n	D1
		HL								21 n n	2A n n	E1
		SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
		IX								DD 21 n n	DD 2A n n	DD E1
		IY								FD 21 n n	FD 2A n n	FD E1
	Extended	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
PUSH Instructions →	Register Indirect	(SP)	F6	C6	D6	E6		DD E6	FD E6			
.												POP Instructions

Note: The PUSH and POP instruction adjust the SP after every execution.

Note: Descriptions of the 16-Bit Load Group instructions begin on page 96.

These 16-bit load operations differ from other 16-bit loads in that the stack pointer is automatically decremented and incremented as each byte is pushed onto or popped from the stack, respectively. For example, the PUSH AF instruction is a single-byte instruction with the op code of `F5h`. During execution, this sequence is generated as:

```
Decrement SP
LD (SP), A
Decrement SP
LD (SP), F
```

The external stack now appears as shown in Figure 36.

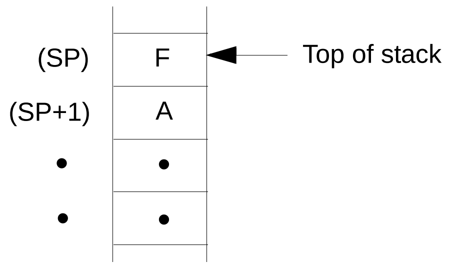


Figure 36. Example of a 16-Bit Load Operation

The POP instruction is the exact reverse of a PUSH. All PUSH and POP instructions utilize a 16-bit operand and the high-order byte is always pushed first and popped last.

```
PUSH BC is PUSH 8 then C
PUSH DE is PUSH D then E
PUSH HL is PUSH H then L
POP HL is POP L then H
```

The instruction using extended immediate addressing for the source requires two bytes of data following the op code, as shown in the following example:

```
LD DE, 0659h
```

The instruction sequence for this value in memory is shown in Figure 37.

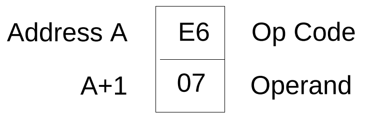


Figure 37. Example of a 2-Byte Load Indexed/Immediate Instruction Sequence

In all extended immediate or extended addressing modes, the low-order byte always appears first after the op code.

Table 7 lists the 16-bit exchange instructions implemented in the Z80 CPU. Op code `08h` allows the programmer to switch between the two pairs of Accumulator flag registers, while `D9h` allows the programmer to switch between the duplicate set of six general-purpose registers. These op codes are only one byte in length to minimize the time necessary to perform the exchange so that the duplicate banks can be used to make fast interrupt response times.

Table 7. Exchanges EX and EXX

		Implied Addressing				
		AF'	**BC', DE', and HL'**	**HL**	**IX**	**IY**
Implied	AF	08				
	BC					
	DE		D9			
	HL					
	DE			EB		
Register Indirect	(SP)			E3	DD E3	FD E3

Block Transfer and Search

Table 8 lists the extremely powerful block transfer instructions. These instructions operate with three registers.

- HL points to the source location
- DE points to the destination location
- BC is a byte counter

Table 8. Block Transfer Group

Destination		Source	
Register Indirect	(DE)	Register Indirect	
		(HL)	
		(ED) A0	LDI – Load (DE) → (HL) Inc HL and DE, Dec BC
		(ED) B0	LDIR, – Load (DE) → (HL) Inc HL and DE, Dec BC; repeat until BC = 0.
		(ED) A8	LDD – Load (DE) → (HL) Inc HL and DE, Dec BC
		(ED) B8	LDDR – Load (DE) → (HL) Dec HL and DE, Dec BC; repeat until BC = 0.

Note: Register HL points to the source; the DE Register points to the destination; the BC Register is a byte counter.

After the programmer initializes these three registers, any of these four instructions can be used. The Load and Increment (LDI) instruction moves one byte from the location pointed to by HL to the location pointed to by DE. Register pairs HL and DE are then automatically incremented and are ready to point to the following locations. The byte counter (i.e., register pair BC) is also decremented at this time. This instruction is valuable when the blocks of data must be moved but other types of processing are required between each move. The Load, Increment and Repeat (LDIR) instruction is an extension of the LDI instruction. The same load and increment operation is repeated until the byte counter reaches the count of zero. As a result, this single instruction can move any block of data from one location to any other.

Because 16-bit registers are used, the size of the block can be up to 64KB long (1KB = 1024 bits) and can be moved from any location in memory to any other location. Furthermore, the blocks can be overlapping because there are no constraints on the data used in the three register pairs.

The LDD and LDDR instructions are similar to LDI and LDIR. The only difference is that register pairs HL and DE are decremented after every move so that a block transfer starts from the highest address of the designated block rather than the lowest.

Table 9 specifies the op codes for the four block search instructions. The first, CPI (Compare and Increment) compares the data in the Accumulator with the contents of the memory location pointed to by Register HL. The result of the compare is stored in one of the flag bits and the HL register pair is then incremented and the byte counter (register pair BC) is decremented.

Table 9. Block Search Group

Search Location	
Register Indirect	
(HL)	
(ED) A1	CPI Inc HL, Dec BC
(ED) B1	CPRI. Inc HL, Dec BC Repeat until) BC = 0 or find match
(ED) A9	WD Dec HL and BC
(ED) B9	CPDR Dec HL and BC Repeat until BC = 0 or find match
Note: HL points to a location in memory to be compared with Accumulator contents; BC is a byte counter.	

Note: Descriptions of the Exchange, Block Transfer, and Search Group instructions begin on page 121.

The CPIR instruction is merely an extension of the CPl instruction in which the compare is repeated until either a match is found or the byte counter (register pair BC) becomes zero. As a result, this single instruction can search the entire memory for any 8-bit character.

The Compare and Decrement (CPD) and Compare, Decrement, and Repeat (CPDR) instructions are similar; however, their only difference is that they decrement HL after every compare so that they search the memory in the opposite direction; i.e., the search is started at the highest location in the memory block.

These block transfer and compare instructions are extremely powerful in string manipulation applications.

Arithmetic and Logical

Table 10 lists all of the 8-bit arithmetic operations that can be performed with the Accumulator. Also listed are the increment (INC) and decrement (DEC) instructions. In all of these instructions, with the exception of INC and DEC, the specified 8-bit operation is performed between the data in the Accumulator and the source data.

Table 10. 8-Bit Arithmetic and Logic

	Source										
	Register Addressing							**Register Indirect**	**Indexed**		**Immediate**
Destination	**A**	**B**	**C**	**D**	**E**	**F**	**L**	**(HL)**	**(IX+d)**	**(IY+d)**	**n**
ADD	87	80	81	82	83	84	85	88	DD 86 d	FD 86 d	C6 n
Add with Carry ADC	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
Subtract SUB	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
Subtract with Carry SBC	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
AND	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
XOR	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
OR	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
Compare CP	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
Increment INC	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
Decrement DEC	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Note: Descriptions of the 8-Bit Arithmetic Group instructions begin on page 142.

The result of the operation is placed in the Accumulator with the exception of the compare (CP) instruction, which leaves the Accumulator unchanged. All of these operations effect the Flag Register as a result of a specified operation.

The INC and DEC instructions specify a register or a memory location as both the source and the destination of the result. When the source operand is addressed using the index registers, the displacement must directly follow. With immediate addressing, the actual operand directly follows. As an example, the AND `07h` instruction is shown in Figure 38.

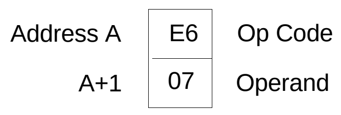


Figure 38. Example of an AND Instruction Sequence

Assuming that the Accumulator contains the value `F3h`, the result of `03h` is placed in the Accumulator:

Accumulator before operation	`1111 0011 = F3h`
Operand	`0000 0111 = 07h`
Result to Accumulator	`0000 0011 = 03h`

The Add (ADD) instruction performs a binary add between the data in the source location and the data in the Accumulator. The Subtract (SUB) instruction performs a binary subtraction. When an Add with Carry (ADC) or Subtract with Carry (SBC) instruction is specified, the Carry flag is also added or subtracted, respectively. The flags and the Decimal Adjust (DAA) instruction in the Z80 CPU allow arithmetic operations for processing the following items:

- Multiprecision packed BCD numbers
- Multiprecision signed or unsigned binary numbers
- Multiprecision two’s complement signed numbers

Other instructions in this group are the Logical And (AND), Logical Or (OR), Exclusive Or (XOR), and Compare (CP) instructions.

Five general-purpose arithmetic instructions operate on the Accumulator or Carry flag. These five instructions are listed in Table 11.

Table 11. General-Purpose AF Operation

Decimal Adjust Accumulator (DAA)	27
Complement Accumulator (CPL)	2F
Negate Accumulator (NEG) (two’s complement	ED 44
Complement Carry Flag (CCF)	3F
Set Carry Flag (SCF)	37

Note: Descriptions of the General-Purpose Arithmetic and CPU Control Groups instructions begin on page 170.

The decimal adjust instruction can adjust for subtraction and addition, making BCD arithmetic operations simple.

Notes:

1. To allow for this operation, the N flag is used. This flag is set if the most recent arithmetic operation was a Subtract. The Negate Accumulator (NEG) instruction forms the two's complement of the number in the Accumulator.
2. A Reset Carry instruction is not included in the Z80 CPU, because this operation can be easily achieved through other instructions such as a logical AND of the Accumulator with itself.

Table 11 lists all of the 16-bit arithmetic operations between 16-bit registers. There are five groups of instructions, including the Add with Carry and Subtract with Carry instructions; ADC and SBC affect all of the flags. These two groups simplify address calculation or other 16-bit arithmetic operations.

Table 12. 16-Bit Arithmetic

		Source					
		BC	DE	HL	SP	IX	IY
Destination	HL	09	19	29	39		
Add (ADD)	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
Add with Carry and set ADC flags	HL	ED 4A	ED 5A	ED 6A	ED 7A		
Subtract with Carry and set SBC flags	HL	ED 42	ED 52	ED 62	ED 72		
Increment (INC)		03	13	23	33	DD 23	FD 23
Decrement (DEC)		DB	1B	2B	3B	DD 2B	FD 2B

Note: Descriptions of the 16-Bit Arithmetic Group instructions begin on page 185.

Rotate and Shift

A major feature of the Z80 CPU is to rotate or shift data in the Accumulator, any general-purpose register, or any memory location. All of the Rotate and Shift op codes are depicted in Figure 39. Also included in the Z80 CPU are arithmetic and logical shift operations. These operations are useful in a wide range of applications including integer multiplication and division. Two BCD digit rotate instructions (RRD and RLD) allow a digit in the Accumulator to be rotated with the two digits in a memory location pointed to by register pair HL. These instructions allow for efficient BCD arithmetic.

Type of Rotate Shift	Source									
	A	B	C	D	E	F	L	(HL)	(IX+d)	(IY+d)
RCL	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 06	CB 0E	DD CB d 06	FD CB d 06
RRC	CB 0F	CB 08	CB 09	CB 0A	CB 06	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
SRA	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
								ED 6F		
								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

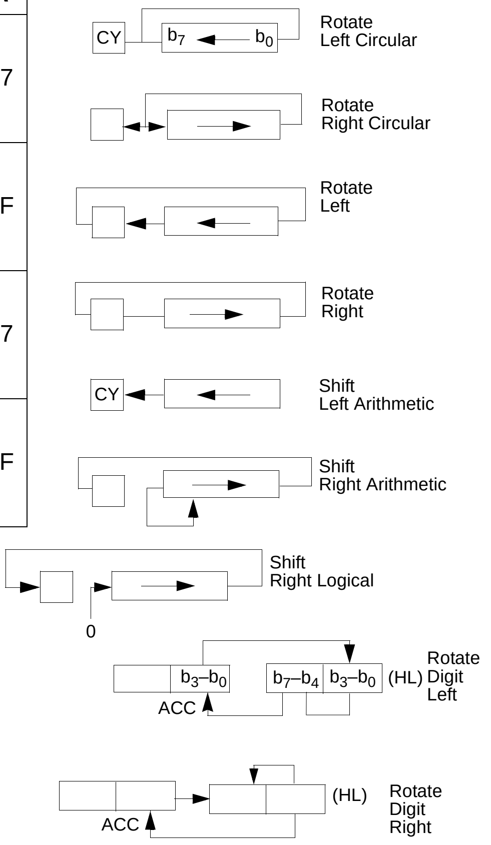


Figure 39. Rotates and Shifts

Note: Descriptions of the Rotate and Shift Group instructions begin on page 202.

Bit Manipulation

The ability to set, reset, and test individual bits in a register or memory location is required in almost every program. These bits can be flags in a general-purpose software routine, indications of external control conditions, or data packed into memory locations, making memory utilization more efficient.

With a single instruction, the Z80 CPU can set, reset, or test any bit in the Accumulator, in any general-purpose register, or in any memory location. Table 13 lists the 240 instructions that are available for this purpose.

Table 13. Bit Manipulation Group

		Register Addressing							Register Indirect	Indexed	
		A	8	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
	Bit									DD	FD
Test Bit	0	C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		47	40	41	42	43	44	45	46	d	d
										46	46
	1									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		4F	48	49	4A	48	4C	4D	4E	d	d
										4E	4E
	2									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		57	50	51	52	53	54	55	56	d	d
										56	56
	3									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		5F	58	59	5A	5B	5C	5D	5E	d	d
										46	46
	4									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		67	60	61	62	63	64	65	66	d	d
										66	66
	5									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		6F	68	69	6A	68	6C	6D	6E	d	d
										6E	6E

Table 13. Bit Manipulation Group (Continued)

		Register Addressing							Register Indirect	Indexed	
Test Bit (cont'd.)	6									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		77	70	71	72	73	74	75	76	d	d
										76	76
	7									DD	DD
		C8	C8	C8	C8	C8	C8	CS	C8	C8	C8
		7F	78	79	7A	78	7C	7D	7E	d	d
										46	46
Rest Bit RES	0									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		87	80	81	82	83	84	85	86	d	d
										86	86
	1									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		8F	88	89	8A	88	8C	8D	8E	d	d
										8E	8E
	2									DD	FD
		C8	C8	CS	C8	C8	C8	C8	C8	C8	C8
		97	90	91	92	93	94	95	96	d	d
										96	96
	3									DD	FD
		C8	C8	C8	C8	CS	C8	C8	C8	C8	C8
		9F	98	99	9A	98	90	90	9E	d	d
										9E	9E
	4									DD	FD
		C8	C8	C8	C8	C6	C8	C8	C8	C8	C8
		A7	AO	AI	A2	A3	A4	A5	A6	d	d
										A6	A6
	5									DD	FD
		C8	C8	C8	C8	08	C8	C8	C8	C8	C8
		AF	A8	A9	AA	AB	AC	AD	AE	d	d
										AE	AE
	6									DD	FD
		C8	C8	C8	C8	C8	C8	C8	C8	C8	C8
		B7	B0	B1	82	B3	B4	B5	B6	d	d
										B6	B6

Table 13. Bit Manipulation Group (Continued)

		Register Addressing							Register Indirect	Indexed	
Rest Bit RES (cont'd.)	7	C8 BF	C8 B8	C8 89	C8 8A	C8 B8	C8 8C	C8 BD	C8 9E	DD C8 d BE	DD C8 d BE
Set Bit SET	0	C8 C7	C8 C0	C8 C1	C8 C2	C8 C3	C8 C4	C8 C5	C8 C6	DD C8 d C6	FD C8 d C6
	1	C8 CF	C8 C8	C8 C9	C8 CA	C8 C8	C8 CC	C8 CD	C8 CE	DD C8 d CE	FD C8 d CE
	2	C8 D7	C8 DO	C8 D1	C8 D2	C8 D3	C8 D4	C8 DS	C8 D6	DD C8 d D6	FD C8 d D6
	3	C8 DF	C8 D8	C8 09	C8 DA	C8 DS	C8 DC	C8 DD	C8 DE	DD C8 d DE	FD C8 d DE
	4	C8 E7	C8 E0	C8 E1	C8 E2	C8 E3	C8 E4	C8 E5	C8 E6	DD C8 d E6	FD C8 d E6
	5	C8 EF	C8 E8	C8 E9	C8 EA	C8 EB	C8 EC	C8 ED	C8 EE	DD C8 d EE	FD C8 d EE
	6	C8 F7	C8 FO	C8 F1	C8 F2	C8 F3	C8 F4	C8 FS	C8 F6	DD C8 d F6	FD C8 d F6
	7	C8 FF	C8 F8	C8 F9	C8 FA	C8 FB	C8 FC	C8 FD	C8 FE	DD C8 d FE	FD C8 d FE

Register addressing can specify the Accumulator or any general-purpose register on which an operation is to be performed. Register Indirect and Indexed addressing are available for operations at external memory locations. Bit test operations set the Zero flag (Z) if the tested bit is a 0.

Note: Descriptions of the Bit Set, Reset, and Test Group instructions begin on page 240.

Jump, Call, and Return

Table 14 lists all of the jump, call, and return instructions implemented in the Z80 CPU. A jump is a branch in a program in which the program counter is loaded with a 16-bit value as specified by one of the three available addressing modes (Immediate Extended, Relative, or Register Indirect). In Table 14, the jump group includes several conditions that can be specified before the jump is made. If these conditions are not met, the program merely continues with the next sequential instruction. The conditions are all dependent on the data in the Flag Register. The immediate extended addressing is used to jump to any location in the memory. This instruction requires three bytes (i.e., two bytes designated to specifying the 16-bit address), with the low-order address byte first, followed by the high-order address byte.

An example of an unconditional jump to memory location `3E32h` is shown in Figure 40.

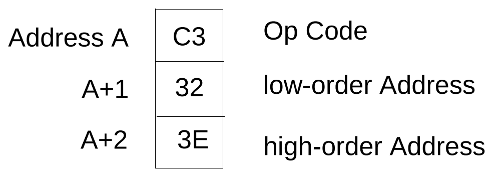


Figure 40. Example of an Unconditional Jump Sequence

The Relative Jump instruction uses only two bytes, the second byte is a signed two's complement displacement from the existing Program Counter. This displacement can be in the range of +129 to –126 and is measured from the address of the instruction op code.

Table 14. Jump, Call, and Return Group

			Condition									
			Un-Cond.	**Carry**	**Non-Carry**	**Zero**	**Non-Zero**	**Parity Even**	**Parity Odd**	**Sign Neg**	**Sign Pos**	**Reg B^{1}0**
JUMP JP	IMMED. EXT.	nn	C3 n n	D8 n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP JR	RELATIVE	PC+e	18 e–2	38 e–2	30 e–2	28 e–2	20 e–2					
JUMP JP	Register INDIR.	(HL)	EB									
		(IX)	DD E9									
		(IY)	FD E9									
CALL	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
Decrement B, Jump If Non-Zero DJNZ	RELATIVE	PC+e										10 e–2
Return RE	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
Return From Interrupt RETI			ED 4D									
Return From Non-Maskable Interrupt RETN			ED 45									

Note: Descriptions of the Jump Group instructions begin on page 258.

Three types of Register Indirect jumps are also included. These instructions are implemented by loading the register pair HL or one of the index registers IX or IY directly into the Program Counter. This feature allows for program jumps to be a function of previous calculations.

A Call is a special form of a jump in which the address of the byte following the call instruction is pushed onto the stack before the jump is made. A return instruction is the reverse of a call because the data on the top of the stack is popped directly into the Program Counter to form a jump address. The call and return instructions allow for simple subroutine and interrupt handling. Two special return instruction are included in the Z80 family of microprocessors. The return from interrupt instruction (RETI) and the return from nonmaskable interrupt (RETN) are treated in the CPU as an unconditional return identical to the op code C9h. The difference is that (RETI) can be used at the end of an interrupt routine and all Z80 peripheral chips recognize the execution of this instruction for proper control of nested priority interrupt handling. This instruction, coupled with the Z80 CPU's peripheral devices implementation, simplifies the normal return from nested interrupt. Without this feature, the following software sequence is necessary to inform the interrupting device that the interrupt routine is completed:

```
Disable Interrupt   ; Prevent interrupt before routine is exited.
LD A, n             ; Notify peripheral that service routine
                    ; is complete.
OUT n, A
Enable Interrupt
Return
```

This seven-byte sequence can be replaced with the one-byte EI instruction and the two-byte RETI instruction in the Z80 CPU. This instruction is important because interrupt service time often must be minimized.

The DJNZ instruction is used to facilitate program loop control. This two-byte relative jump instruction decrements Register B, and the jump occurs if Register B is not decremented to 0. The relative displacement is expressed as a signed two's complement number. A simple example of its use is shown in Table 15.

Table 15. Example Usage of the DJNZ Instruction

Address	Instruction	Comments
N, N+1	LD B, 7	; Set B Register to count of 7
N+2 to N+9	(Perform a sequence of instructions)	; Loop to be performed 7 times
N+10,N+11	DJNZ – 8	; To jump from N+12 to N+2
N + 12	(Next instruction)	

Table 16 lists the eight op codes for the Restart instruction, which is a single-byte call to any of the eight addresses listed. A simple mnemonic for each of these eight calls is also listed. The Restart instruction is useful for frequently-used routines due to its minimal memory consumption.

Table 16. Restart Group

		Op Code	
CALL Address	0000h	C7	RST 0
	0008h	CF	RST 8
	0010h	D7	RST 16
	0018h	DF	RST 24
	0020h	E7	RST 32
	0028h	EF	RST 40
	0030h	F7	RST 48
	0038h	FF	RST 56

Note: Descriptions of the Call and Return Group instructions begin on page 277.

Input/Output

The Z80 CPU contains an extensive set of input and output instructions, as shown in Tables 17 and 18. The addressing of the input or output device can be either absolute or Register Indirect, using the C register. In the Register Indirect addressing mode, data can be transferred between the I/O devices and any of the internal registers. In addition, eight block transfer instructions are implemented. These instructions are similar to the memory block transfers except that they use register pair HL for a pointer to the memory source (output commands) or destination (input commands) while Register B is used as a byte counter. Register C holds the address of the port for which the input or output command is required. Because Register B is eight bits in length, the I/O block transfer command handles up to 256 bytes.

In the IN A and OUT n, A instructions, the I/O device's *n* address appears in the lower half of the address bus (A7–A0), while the Accumulator content is transferred in the upper half of the address bus. In all Register Indirect input output instructions, including block I/O transfers, the contents of the C Register are transferred to the lower half of the address bus (device address) while the contents of Register B are transferred to the upper half of the address bus.

Table 17. Input Group

				Immediate (n)	Register Indirect) (c)	
Input Destination	Input IN	Register Address	A	DB n	ED 7B	
			B		ED 40	
			C		ED 48	
			D		ED 50	
			E		ED 58	
			H		ED 60	
			L		ED 68	
	INI: input & inc HL, Dec B	Register Indir	(HL)		ED A2	Block Input Commands
	INIR: INP, Inc HL, Dec B, repeat if B≠0				ED B2	
	IND: input & Inc Dec HL, Dec B				ED AA	
	INDR: input, Dec HL, Dec B, repeat if B≠0				ED BA	

Table 18. 8-Bit Arithmetic and Logic

Source

			Register							**Register Indirect**	
			A	**B**	**C**	**D**	**E**	**H**	**L**	**(HL)**	
11OUT	Immediate	(n)	D3 n								
11OUT	Register Indirect	(c)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69		
11OUT: output inc HL, dec B	Register Indirect	(c)								ED A3	Block Output Command
11OUT: output dec B, repeat if B≠0	Register Indirect	(c)								ED B3	
11OUT: output dec HL and B	Register Indirect	(c)								ED AB	
11OUTDR: output, dec HL and B, repeat if B≠0	Register Indirect	(c)								ED BB	
	Port Destination Address										

Note: Descriptions of the Input and Output Group instructions begin on page 291.

CPU Control Group

Table 19 shows the six general-purpose CPU control instructions. The HALT instruction suspends CPU operation until a subsequent interrupt is received, while the DI and EI are used to lock out and enable interrupts. The three interrupt mode commands set the CPU to any of the three available interrupt response modes; each of these is described in the next paragraph. The NOP instruction has no function.

Table 19. Miscellaneous CPU Control

NOP	00	
HALT	76	
Disable INT (EI)	F3	
Enable INT (EI)	FB	
Set INT Mode 0 IM0	ED 46	8080A mode
Set INT Mode 1 IM1	ED 56	Call to address 0038h
Set INT Mode 2 IM2	ED 5E	Indirect call using Register I and B bits from INTER device as a pointer

If Mode 0 is set, the interrupting device can insert any instruction on the data bus and allow the CPU to execute it. Mode 1 is a simplified mode in which the CPU automatically executes a restart (RST) at address `0038h` so that no external hardware is required (the old Program Counter content is pushed onto the stack). Mode 2 is the most powerful because it allows for an indirect call to any location in memory. With this mode, the CPU forms a 16-bit memory address in which the upper eight bits are the contents of Register I, and the lower eight bits are supplied by the interrupting device. This address points to the first of two sequential bytes in a table in which the address of the service routine is located, as shown in Figure 41. The CPU automatically obtains the starting address and performs a CALL instruction to this address.

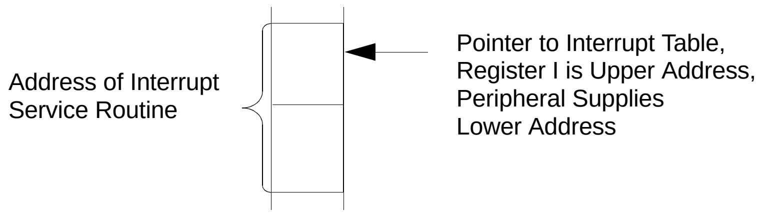


Figure 41. Mode 2 Interrupt Command

Z80 Instruction Set

This chapter provides a description of the assembly language instructions available with the Z80 CPU.

Z80 Assembly Language

Assembly language allows the user to write a program without concern for memory addresses or machine instruction formats. It uses symbolic addresses to identify memory locations and mnemonic codes (op codes and operands) to represent the instructions. Labels (symbols) are assigned to a particular instruction step in a source program to identify that step as an entry point for use in subsequent instructions. Operands following each instruction represent storage locations, registers, or constant values. The assembly language also includes assembler directives that supplement the machine instruction. A pseudo-op, for example, is a statement that is not translated to a machine instruction, but rather is interpreted as a directive that controls the assembly process.

A program written in assembly language is called a *source program*, which consists of symbolic commands called *statements*. Each statement is written on a single line and can consist of one to four entries: A label field, an operation field, an operand field, and a comment field. The source program is processed by the assembler to obtain a machine language program (object program) that can be executed directly by the Z80 CPU.

Zilog provides several assemblers that differ in the features offered. Both absolute and relocatable assemblers are available with the Development and Micro-computer Systems. The absolute assembler is contained in base level software operating in a 16K memory space, while the relocating assembler is part of the RIO environment operating in a 32K memory space.

Z80 Status Indicator Flags

The Flag registers, F and F', supply information to the user about the status of the Z80 CPU at any particular time. The bit positions for each flag are listed in Table 20 and defined in

Table 20. Flag Register Bit Positions

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Position	S	Z	X	H	X	P/V	N	C

Table 21. Flag Definitions

Symbol	Field Name
C	Carry Flag
N	Add/Subtract
P/V	Parity/Overflow Flag
H	Half Carry Flag
Z	Zero Flag
S	Sign Flag
X	Not Used

Each of these two Flag registers contains 6 bits of status information that are set or cleared by CPU operations; bits 3 and 5 are not used. Four of these bits (C, P/V, Z, and S) can be tested for use with conditional JUMP, CALL, or RETURN instructions. The H and N flags cannot be tested; these two flags are used for BCD arithmetic.

Carry Flag

The Carry Flag (C) is set or cleared depending on the operation being performed. For ADD instructions that generate a Carry, and for SUB instructions that generate a Borrow, the Carry Flag is set. The Carry Flag is reset by an ADD instruction that does not generate a Carry, and by a SUB instruction that does not generate a Borrow. This saved Carry facilitates software routines for extended precision arithmetic. Additionally, the DAA instruction sets the Carry Flag if the conditions for making the decimal adjustment are met.

For the RLA, RRA, RLS, and RRS instructions, the Carry bit is used as a link between the least-significant byte (LSB) and the most-significant byte (MSB) for any register or memory location. During the RLCA, RLC, and SLA instructions, the Carry flag contains the final value shifted out of bit 7 of any register or memory location. During the RRCA, RRC, SRA, and SRL instructions, the Carry flag contains the final value shifted out of bit 0 of any register or memory location.

For the logical instructions AND, OR, and XOR, the Carry flag is reset.

The Carry flag can also be set by the Set Carry Flag (SCF) instruction and complemented by the Compliment Carry Flag (CCF) instruction.

Add/Subtract Flag

The Add/Subtract Flag (N) is used by the Decimal Adjust Accumulator instruction (DAA) to distinguish between the ADD and SUB instructions. For ADD instructions, N is cleared to 0. For SUB instructions, N is set to 1.

Decimal Adjust Accumulator Flag

The Decimal Adjust Accumulator (DAA) instruction uses this flag to distinguish between ADD and SUBTRACT instructions. For all ADD instructions, N sets to 0. For all SUBTRACT instructions, N sets to 1.

Parity/Overflow Flag

The Parity/Overflow (P/V) Flag is set to a specific state depending on the operation being performed. For arithmetic operations, this flag indicates an overflow condition when the result in the Accumulator is greater than the maximum possible number (+127) or is less than the minimum possible number (–128). This overflow condition is determined by examining the sign bits of the operands.

For addition, operands with different signs never cause overflow. When adding operands with similar signs and the result contains a different sign, the Overflow Flag is set, as shown in the following example.

```
+120    =  0111    1000    ADDEND
+105    =  0110    1001    AUGEND
                                  ____
+225    =  1110    0001    (–95)   SUM
```

The two numbers added together result in a number that exceeds +127 and the two positive operands result in a negative number (–95), which is incorrect. The Overflow Flag is therefore set.

For subtraction, overflow can occur for operands of unalike signs. Operands of alike signs never cause overflow, as shown in the following example.

```
      +127   0111      1111   MINUEND
  (–)  –64   1100      0000   SUBTRAHEND
--------------------------------------------
      +191   1011      1111   DIFFERENCE
```

The minuend sign has changed from a positive to a negative, resulting in an incorrect difference; the Overflow Flag is set.

Another method for identifying an overflow is to observe the Carry to and out of the sign bit. If there is a Carry in and no Carry out, or if there is no Carry in and a Carry out, then an Overflow has occurred.

This flag is also used with logical operations and rotate instructions to indicate the resulting parity is even. The number of 1 bits in a byte are counted. If the total is Odd, ODD parity is flagged (i.e., P = 0). If the total is even, even parity is flagged (i.e., P = 1).

During the CPI, CPIR, CPD, and CPDR search instructions and the LDI, LDIR, LDD, and LDDR block transfer instructions, the P/V Flag monitors the state of the Byte Count (BC) Register. When decrementing, if the byte counter decrements to 0, the flag is cleared to 0; otherwise the flag is set to1.

During the LD *A, I* and LD *A, R* instructions, the P/V Flag is set with the value of the interrupt enable flip-flop (IFF2) for storage or testing.

When inputting a byte from an I/O device with an IN *r, (C)* instruction, the P/V Flag is adjusted to indicate data parity.

Half Carry Flag

The Half Carry Flag (H) is set (1) or cleared (0) depending on the Carry and Borrow status between bits 3 and 4 of an 8-bit arithmetic operation. This flag is used by the Decimal Adjust Accumulator (DAA) instruction to correct the result of a packed BCD add or subtract operation. The H Flag is set (1) or cleared (0) as shown in Table 22.

Table 22. Half Carry Flag Add/Subtract Operations

H Flag	Add	Subtract
1	A Carry occurs from bit 3 to bit 4	A Borrow from bit 4 occurs
0	No Carry occurs from bit 3 to bit 4	No Borrow from bit 4 occurs

Zero Flag

The Zero Flag (Z) is set (1) or cleared (0) if the result generated by the execution of certain instructions is 0.

For 8-bit arithmetic and logical operations, the Z flag is set to a 1 if the resulting byte in the Accumulator is 0. If the byte is not 0, the Z flag is reset to 0.

For Compare (search) instructions, the Z flag is set to 1 if the value in the Accumulator is equal to the value in the memory location indicated by the value of the register pair HL.

When testing a bit in a register or memory location, the Z flag contains the complemented state of the indicated bit (see *Bit b, r* in the Bit Set, Reset, and Test Group section on page 240).

When inputting or outputting a byte between a memory location and an INI, IND, OUTI, or OUTD I/O device, if the result of decrementing Register B is 0, then the Z flag is 1; otherwise, the Z flag is 0. Additionally, for byte inputs from I/O devices using IN *r, (C)*, the Z flag is set to indicate a 0-byte input.

Sign Flag

The Sign Flag (S) stores the state of the most-significant bit of the Accumulator (bit 7). When the Z80 CPU performs arithmetic operations on signed numbers, the binary twos-complement notation is used to represent and process numeric information. A positive number is identified by a 0 in Bit 7. A negative number is identified by a 1. The binary equivalent of the magnitude of a positive number is stored in bits 0 to 6 for a total range of from 0 to 127. A negative number is represented by the twos complement of the equivalent positive number. The total range for negative numbers is from –1 to –128.

When inputting a byte from an I/O device to a register using an IN *r, (C)* instruction, the S Flag indicates either positive (S = 0) or negative (S = 1) data.

Z80 Instruction Description

Execution time (E.T.) for each instruction is provided in microseconds for an assumed 4MHz clock. Total machine cycles (M) are indicated with total clock periods, or *T states*. Also indicated are the number of T states for each M cycle, as shown in the following example.

M Cycles	T States	E.T.
2	7(4,3) 4 MHz	1.75

This example indicates that the instruction consists of two machine cycles. The first cycle contains 4 clock periods/T states). The second cycle contains 3 clock periods, for a total of 7 clock periods/T states. The instruction executes in 1.75 microseconds.

In the register format of each of the instructions that follow, the most-significant bit to the left and the least-significant bit to the right.

8-Bit Load Group

The following 8-bit load instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

LD r, r'

Operation

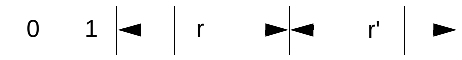
r, ← r′

Op Code

LD

Operands

r, r′



Description

The contents of any register *r′* are loaded to any other register *r. r, r′* identifies any of the registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r, C
A	`111`
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`

M Cycles	T States	MHz E.T.
1	4	1.0

Condition Bits Affected

None.

Example

If the H Register contains the number `8Ah`, and the E register contains `10h`, the instruction LD H, E results in both registers containing `10h`.

LD r,n

Operation

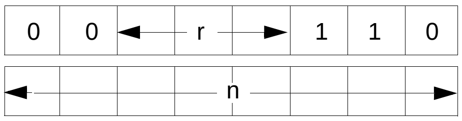
r ← n

Op Code

LD

Operands

r, *n*



Description

The 8-bit integer *n* is loaded to any register *r*, in which *r* identifies registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

Upon the execution of an LD *E*, `A5h` instruction, Register E contains `A5h`.

LD r, (HL)

Operation

r ← (HL)

Op Code

LD

Operands

r, (HL)

0	1	◄	r	►	1	1	0

Description

The 8-bit contents of memory location (HL) are loaded to register *r*, in which *r* identifies registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

If register pair HL contains the number `75A1h`, and memory address `75A1h` contains byte `58h`, the execution of LD C, *(HL)* results in `58h` in Register C.

LD r, (IX+d)

Operation

r ← (IX+d)

Op Code

LD

Operands

r, (IX+d)

1	1	0	1	1	1	0	1	DD
0	1	◄—	r	—►	1	1	0	
◄—			d				—►	

Description

The *(IX+d)* operand (i.e., the contents of Index Register IX summed with two's-complement displacement integer *d*) is loaded to register *r*, in which *r* identifies registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	2.50

Condition Bits Affected

None.

Example

If Index Register IX contains the number `25AFh`, the instruction LD B, (IX+19h) allows the calculation of the sum `25AFh` + `19h`, which points to memory location `25C8h`. If this address contains byte `39h`, the instruction results in Register B also containing `39h`.

LD r, (IY+d)

Operation

r ← (IY+D)

Op Code

LD

Operands

r, (lY+d)

1	1	1	1	1	1	1	1	FD
0	1	◄	r	►	1	1	0	
◄			d				►	

Description

The operand *(lY+d)* loads the contents of Index Register IY summed with two's-complement displacement integer, *d*, to register *r*, in which *r* identifies registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

None.

Example

If Index Register IY contains the number `25AFh`, the instruction LD B, (IY+19h) allows the calculation of the sum `25AFh` + `19h`, which points to memory location `25C8h`. If this address contains byte `39h`, the instruction results in Register B also containing `39h`.

LD (HL), r

Operation

(HL) ← r

Op Code

LD

Operands

(HL), r

0	1	1	1	0	←	r	→

Description

The contents of register *r* are loaded to the memory location specified by the contents of the HL register pair. The *r* symbol identifies registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

If the contents of register pair HL specify memory location `2146h` and Register B contains byte `29h`, then upon the execution of an LD *(HL), B* instruction, memory address `2146h` also contains `29h`.

LD (IX+d), r

Operation

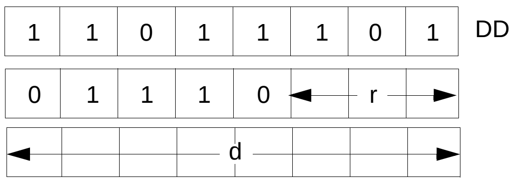
(IX+d) ← r

Op Code

LD

Operands

(IX+d), r



Description

The contents of register *r* are loaded to the memory address specified by the contents of Index Register IX summed with *d*, a two's-complement displacement integer. The *r* symbol identifies registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

None.

Example

If the C register contains byte `1Ch`, and Index Register IX contains `3100h`, then the instruction LID *(IX +* `6h`*), C* performs the sum `3100h` + `6h` and loads `1Ch` to memory location `3106h`.

LD (IY+d), r

Operation

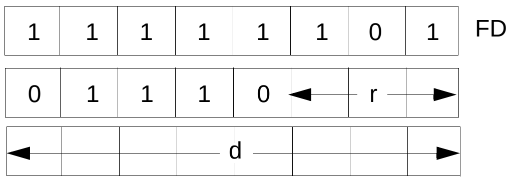
(lY+d) ← r

Op Code

LD

Operands

(lY+d), r



Description

The contents of resister *r* are loaded to the memory address specified by the sum of the contents of Index Register IY and *d*, a two's-complement displacement integer. The *r* symbol is specified according to the following table.

Register	**r**
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	**T States**	**4 MHz E.T.**
5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

None.

Example

If the C register contains byte `48h`, and Index Register IY contains `2A11h`, then the instruction LD *(IY* + `4h`*), C* performs the sum `2A11h` + `4h`, and loads `48h` to memory location `2A15`.

LD (HL), n

Operation

(HL) ← n

Op Code

LD

Operands

(HL), n

0	0	1	1	0	1	1	0	36
◄			n				►	

Description

The *n* integer is loaded to the memory address specified by the contents of the HL register pair.

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

Example

If the HL register pair contains `4444h`, the instruction LD *(HL)*, `28h` results in the memory location `4444h` containing byte `28h`.

LD (IX+d), n

Operation

(IX+d) ← n

Op Code

LD

Operands

(IX+d), n

1	1	0	1	1	1	0	1	DD
0	0	1	1	0	1	1	0	36
◄				d			►	
◄				n			►	

Description

The *n* operand is loaded to the memory address specified by the sum of Index Register IX and the two's complement displacement operand *d*.

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3,5,3)	4.75

Condition Bits Affected

None.

Example

If Index Register IX contains the number `219Ah`, then upon execution of an LD *(IX+*`5h`*)*, `5Ah` instruction, byte `5Ah` is contained in memory address `219Fh`.

LD (IY+d), n

Operation

(lY+d) ← n

Op Code

LD

Operands

(lY+d), n

1	1	1	1	1	1	0	1	FD
0	0	1	1	0	1	1	0	36
◄			d				►	
◄			n				►	

Description

The *n* integer is loaded to the memory location specified by the contents of Index Register summed with the two's-complement displacement integer, *d*.

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	2.50

Condition Bits Affected

None.

Example

If Index Register IY contains the number `A940h`, the instruction LD (IY+10h), `97h` results in byte `97h` in memory location `A950h`.

LD A, (BC)

Operation

A ← (BC)

Op Code

LD

Operands

A, (BC)

0	0	0	0	1	0	1	0	0A

Description

The contents of the memory location specified by the contents of the BC register pair are loaded to the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

If the BC register pair contains the number `4747h`, and memory address `4747h` contains byte `12h`, then the instruction LD *A, (BC)* results in byte `12h` in Register A.

LD A, (DE)

Operation

A ← (DE)

Op Code

LD

Operands

A, (DE)

0	0	0	1	1	0	1	0	1A

Description

The contents of the memory location specified by the register pair DE are loaded to the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

If the DE register pair contains the number `30A2h` and memory address `30A2h` contains byte `22h`, then the instruction LD *A, (DE)* results in byte `22h` in Register A.

LD A, (nn)

Operation

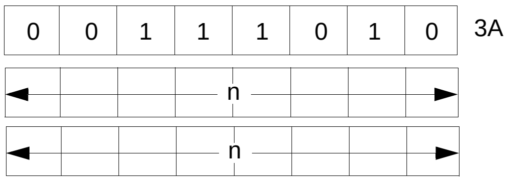
A ← (nn)

Op Code

LD

Operands

A, (*nn*)



Description

The contents of the memory location specified by the operands *nn* are loaded to the Accumulator. The first *n* operand after the op code is the low-order byte of a 2-byte memory address.

M Cycles	T States	4 MHz E.T.
4	13 (4, 3, 3, 3)	3.25

Condition Bits Affected

None.

Example

If *nn* contains `8832h` and memory address `8832h` contains byte `04h`, then upon the execution of an LD *A*, (*nn*) instruction, the `04h` byte is in the Accumulator.

LD (BC), A

Operation

(BC) ← A

Op Code

LD

Operands

(BC), A

0	0	0	0	0	0	1	0	02

Description

The contents of the Accumulator are loaded to the memory location specified by the contents of the register pair BC.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

If the Accumulator contains `7Ah` and the BC register pair contains `1212h` the instruction LD *(BC), A* results in `7Ah` in memory location `1212h`.

LD (DE), A

Operation

(DE) ← A

Op Code

LD

Operands

(DE), A

0	0	0	1	0	0	1	0	12

Description

The contents of the Accumulator are loaded to the memory location specified by the contents of the DE register pair.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

If register pair DE contains `1128h` and the Accumulator contains byte `A0h`, then the execution of a LD *(DE), A* instruction results in `A0h` being stored in memory location `1128h`.

LD (nn), A

Operation

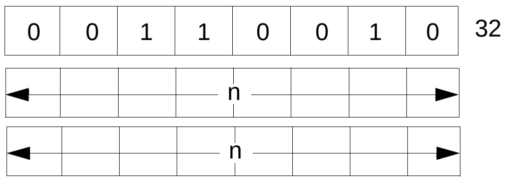
(nn) ← A

Op Code

LD

Operands

(*nn*), *A*



Description

The contents of the Accumulator are loaded to the memory address specified by the operand *nn*. The first *n* operand after the op code is the low-order byte of *nn*.

M Cycles	T States	4 MHz E.T.
4	13 (4, 3, 3, 3)	3.25

Condition Bits Affected

None.

Example

If the Accumulator contains byte `D7h`, then executing an LD (`3141h`), `AD7h` instruction results in memory location `3141h`.

LD A, I

Operation

A ← 1

Op Code

LD

Operands

A, *I*

1	1	1	0	1	1	0	1	ED
0	1	0	1	0	1	1	1	57

Description

The contents of the Interrupt Vector Register I are loaded to the Accumulator.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

Condition Bits Affected

S is set if the I Register is negative; otherwise, it is reset.

Z is set if the I Register is 0; otherwise, it is reset.

H is reset.

P/V contains contents of IFF2.

N is reset.

C is not affected.

If an interrupt occurs during execution of this instruction, the Parity flag contains a 0.

LD A, R

Operation

A ← R

Op Code

LD

Operands

A, *R*

1	1	1	0	1	1	0	1	ED
0	1	0	1	1	1	1	1	5F

Description

The contents of Memory Refresh Register R are loaded to the Accumulator.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

Condition Bits Affected

S is set if, R-Register is negative; otherwise, it is reset.

Z is set if the R Register is 0; otherwise, it is reset.

H is reset.

P/V contains contents of IFF2.

N is reset.

C is not affected.

If an interrupt occurs during execution of this instruction, the parity flag contains a 0.

LD I,A

Operation

I ← A

Op Code

LD

Operands

I, A

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	1	1	47

Description

The contents of the Accumulator are loaded to the Interrupt Control Vector Register, I.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

Condition Bits Affected

None.

LD R, A

Operation

R ← A

Op Code

LD

Operands

R, A

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	1	1	4F

Description

The contents of the Accumulator are loaded to the Memory Refresh register R.

M Cycles	T States	MHz E.T.
2	9 (4, 5)	2.25

Condition Bits Affected

None.

16-Bit Load Group

The following 16-bit load instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

LD dd, nn

Operation

dd ← nn

Op Code

LD

Operands

dd, nn

0	0	d	d	0	0	0	1
◄			n				►
◄			n				►

Description

The 2-byte integer *nn* is loaded to the *dd* register pair, in which *dd* defines the BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	`00`
DE	`01`
HL	`10`
SP	`11`

The first *n* operand after the op code is the low-order byte.

M Cycles	T States	4 MHz E.T.
2	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

Example

Upon the execution of an LD *HL*, `5000h` instruction, the HL register pair contains `5000h`.

LD IX, nn

Operation

IX ← nn

Op Code

LD

Operands

IX, nn

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	0	1	21
◄			n				►	
◄			n				►	

Description

The *n* integer is loaded to Index Register IX. The first *n* operand after the op code is the low-order byte.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected

None.

Example

Upon the execution of an LD *IX*, `45A2h` instruction, the index register contains integer `45A2h`.

LD IY, nn

Operation

IY ← nn

Op Code

LD

Operands

IY, nn

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	0	1	21
◄			n				►	
◄			n				►	

Description

The *nn* integer is loaded to Index Register IY. The first *n* operand after the op code is the low-order byte.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected

None.

Example

Upon the execution of a LD *IY,* `7733h` instruction, Index Register IY contains the integer `7733h`.

LD HL, (nn)

Operation

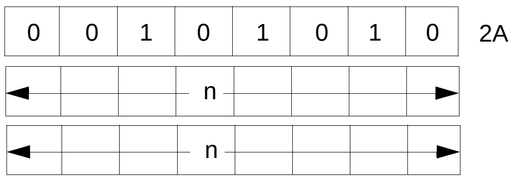
H ← (nn+1), L ← (nn)

Op Code

LD

Operands

HL, (*nn*)



Description

The contents of memory address (*nn*) are loaded to the low-order portion of register pair HL (Register L), and the contents of the next highest memory address (*nn*+1) are loaded to the high-order portion of HL (Register H). The first *n* operand after the op code is the low-order byte of *nn*.

M Cycles	T States	4 MHz E.T.
5	16 (4, 3, 3, 3, 3)	4.00

Condition Bits Affected

None.

Example

If address `4545h` contains `37h` and address `4546h` contains `A1h`, then upon the execution of an LD *HL*, *(*`4545h`*)* instruction, the HL register pair contains `A137h`.

LD dd, (nn)

Operation

ddh ← (nn+1) ddl ← (nn)

Op Code

LD

Operands

dd, (*nn*)

1	1	1	0	1	1	0	1	ED
0	1	d	d	1	0	1	1	
◄			n				►	
◄			n				►	

Description

The contents of address (*nn*) are loaded to the low-order portion of register pair *dd*, and the contents of the next highest memory address (*nn*+1) are loaded to the high-order portion of *dd*. Register pair *dd* defines BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	
DE	01
HL	10
SP	11

The first *n* operand after the op code is the low-order byte of (*nn*).

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected

None.

Example

If Address `2130h` contains `65h` and address `2131h` contains `78h`, then upon the execution of an LD *BC*, *(*`2130h`*)* instruction, the BC register pair contains `7865h`.

LD IX, (nn)

Operation

IXh ← (nn+1), IXI ← (nn)

Op Code

LD

Operands

IX, (*nn*)

1	1	0	1	1	1	0	1	DD
0	0	1	0	1	0	1	0	2A
◄				n			►	
◄				n			►	

Description

The contents of the address (*nn*) are loaded to the low-order portion of Index Register IX, and the contents of the next highest memory address (*nn*+1) are loaded to the high-order portion of IX. The first *n* operand after the op code is the low-order byte of *nn*.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected

None.

Example

If address `6666h` contains `92h`, and address `6667h` contains `DAh`, then upon the execution of an LD *IX*, *(*`6666h`*)* instruction, Index Register IX contains `DA92h`.

LD IY, (nn)

Operation

IYh ← (nn+1), IYI ← nn)

Op Code

LD

Operands

IY, (*nn*)

1	1	1	1	1	1	0	1	FD
0	0	1	0	1	0	1	0	2A
◄			n				►	
◄			n				►	

Description

The contents of address (*nn*) are loaded to the low-order portion of Index Register IY, and the contents of the next highest memory address (*nn*+1) are loaded to the high-order portion of IY. The first *n* operand after the op code is the low-order byte of *nn*.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected

None.

Example

If address `6666h` contains `92h`, and address `6667h` contains `DAh`, then upon the execution of an LD *IY, (*`6666h`*)* instruction, Index Register IY contains `DA92h`.

LD (nn), HL

Operation

(nn+1) ← H, (nn) ← L

Op Code

LD

Operands

(*nn*), *HL*

0	0	1	0	0	0	1	0	22
◄			n				►	
◄			n				►	

Description

The contents of the low-order portion of register pair HL (Register L) are loaded to memory address (*nn*), and the contents of the high-order portion of HL (Register H) are loaded to the next highest memory address (*nn*+1). The first *n* operand after the op code is the low-order byte of *nn*.

M Cycles	T States	4 MHz E.T.
5	16 (4, 3, 3, 3, 3)	4.00

Condition Bits Affected

None.

Example

If register pair HL contains 483Ah, then upon the execution of an LD *(B2291 – 1), HL* instruction, address B229h contains 3Ah and address B22Ah contains 48h.

LD (nn), dd

Operation

(nn+1) ← ddh, (nn) ← ddl

Op Code

LD

Operands

(*nn*), *dd*

1	1	1	0	1	1	0	1	ED
0	1	d	d	0	0	1	1	
◄				n			►	
◄				n			►	

Description

The low-order byte of register pair *dd* is loaded to memory address (*nn*); the upper byte is loaded to memory address (*nn*+1). Register pair *dd* defines either BC, DE, HL, or SP, assembled as follows in the object code:

Pair	**dd**
BC	00
DE	01
HL	10
SP	11

The first *n* operand after the op code is the low-order byte of a two byte memory address.

M Cycles	**T States**	**4 MHz E.T.**
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected

None.

Example

If register pair BC contains the number `4644h`, the instruction `LD (1000h)`, BC results in `44h` in memory location `1000h`, and `46h` in memory location `1001h`.

LD (nn), IX

Operation

(nn+1) ← IXh, (nn) ← IXI

Op Code

LD

Operands

(*nn*), *IX*

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	1	0	22
◄			n				►	
◄			n				►	

Description

The low-order byte in Index Register IX is loaded to memory address (*nn*); the upper order byte is loaded to the next highest address (*nn*+1). The first *n* operand after the op code is the low-order byte of *nn*.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected

None.

Example

If Index Register IX contains `5A30h`, then upon the execution of an LD *(*`4392h`*), IX* instruction, memory location `4392h` contains number `30h` and location `4393h` contains `5Ah`.

LD (nn), IY

Operation

(nn+1) ← IYh, (nn) ← IYI

Op Code

LD

Operands

(*nn*), *IY*

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	1	0	22
◄				n			►	
◄				n			►	

Description

The low-order byte in Index Register IY is loaded to memory address (*nn*); the upper order byte is loaded to memory location (*nn*+1). The first *n* operand after the op code is the low-order byte of *nn*.

M Cycles	T States	4 MHz E.T.
6	20 (4, 4, 3, 3, 3, 3)	5.00

Condition Bits Affected

None.

Example

If Index Register IY contains `4174h`, then upon the execution of an LD *(*`8838h`*), IY* instruction, memory location `8838h` contains `74h` and memory location `8839h` contains `41h`.

LD SP, HL

Operation

SP ← HL

Op Code

LD

Operands

SP, HL

1	1	1	r	1	0	0	1	F9

Description

The contents of the register pair HL are loaded to the Stack Pointer (SP).

M Cycles	T States	4 MHz E.T.
1	6	1.5

Condition Bits Affected

None.

Example

If the register pair HL contains `442Eh`, then upon the execution of an LD *SP, HL* instruction, the Stack Pointer also contains `442Eh`.

LD SP, IX

Operation

SP ← IX

Op Code

LD

Operands

SP, IX

1	1	0	1	1	1	0	1	DD
1	1	1	1	1	0	0	1	F9

Description

The 2-byte contents of Index Register IX are loaded to the Stack Pointer (SP).

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected

None.

Example

If Index Register IX contains `98DAh`, then upon the execution of an LD *SP, IX* instruction, the Stack Pointer also contains `98DAh`.

LD SP, IY

Operation

SP ← IY

Op Code

LD

Operands

SP, IY

1	1	1	1	1	1	0	1	FD
1	1	1	1	1	0	0	1	F9

Description

The 2-byte contents of Index Register IY are loaded to the Stack Pointer SP.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected

None.

Example

If Index Register IY contains the integer `A227h`, then upon the execution of an LD *SP, IY* instruction, the Stack Pointer also contains `A227h`.

PUSH qq

Operation

(SP – 2) ← qqL, (SP – 1) ← qqH

Op Code

PUSH

Operand

qq

1	1	q	q	0	1	0	1

Description

The contents of the register pair *qq* are pushed to the external memory last-in, first-out (LIFO) stack. The Stack Pointer (SP) Register pair holds the 16-bit address of the current top of the Stack. This instruction first decrements SP and loads the high-order byte of register pair *qq* to the memory address specified by the SP. The SP is decremented again and loads the low-order byte of *qq* to the memory location corresponding to this new address in the SP. The operand *qq* identifies register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	qq
BC	00
DE	01
HL	10
AF	11

M Cycles	T States	4 MHz E.T.
3	11 (5, 3, 3)	2.75

Condition Bits Affected

None.

Example

If the AF Register pair contains `2233h` and the Stack Pointer contains `1007h`, then upon the execution of a PUSH *AF* instruction, memory address `1006h` contains `22h`, memory address `1005h` contains `33h`, and the Stack Pointer contains `1005h`.

PUSH IX

Operation

(SP – 2) ← IXL, (SP – 1) ← IXH

Op Code

PUSH

Operand

IX

1	1	0	1	1	1	0	1	DD
1	1	1	0	0	1	0	1	E5

Description

The contents of Index Register IX are pushed to the external memory last-in, first-out (LIFO) stack. The Stack Pointer (SP) Register pair holds the 16-bit address of the current top of the Stack. This instruction first decrements SP and loads the high-order byte of IX to the memory address specified by SP; then decrements SP again and loads the low-order byte to the memory location corresponding to this new address in SP.

M Cycles	T States	4 MHz E.T.
4	15 (4, 5, 3, 3)	3.75

Condition Bits Affected

None.

Example

If Index Register IX contains `2233h` and the Stack Pointer contains `1007h`, then upon the execution of a PUSH *IX* instruction, memory address `1006h` contains `22h`, memory address `1005h` contains `33h`, and the Stack Pointer contains `1005h`.

PUSH IY

Operation

(SP – 2) ← IYL, (SP – 1) ← IYH

Op Code

PUSH

Operand

IY

1	1	1	1	1	1	0	1	FD
1	1	1	0	0	1	0	1	E5

Description

The contents of Index Register IY are pushed to the external memory last-in, first-out (LIFO) stack. The Stack Pointer (SP) Register pair holds the 16-bit address of the current top of the Stack. This instruction first decrements the SP and loads the high-order byte of IY to the memory address specified by SP; then decrements SP again and loads the low-order byte to the memory location corresponding to this new address in SP.

M Cycles	T States	4 MHz E.T.
4	15 (4, 5, 3, 3)	3.75

Condition Bits Affected

None.

Example

If Index Register IY contains `2233h` and the Stack Pointer contains `1007h`, then upon the execution of a PUSH *IY* instruction, memory address `1006h` contains `22h`, memory address `1005h` contains `33h`, and the Stack Pointer contains `1005h`.

POP qq

Operation

qqH ← (SP+1), qqL ← (SP)

Op Code

POP

Operand

qq

1	1	q	q	0	0	0	1

Description

The top two bytes of the external memory last-in, first-out (LIFO) stack are popped to register pair *qq*. The Stack Pointer (SP) Register pair holds the 16-bit address of the current top of the Stack. This instruction first loads to the low-order portion of *qq*, the byte at memory location corresponding to the contents of SP; then SP is incriminated and the contents of the corresponding adjacent memory location are loaded to the high-order portion of *qq* and the SP is now incriminated again. The operand *qq* identifies register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	r
BC	00
DE	01
HL	10
AF	11

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

Example

If the Stack Pointer contains `1000h`, memory location `1000h` contains `55h`, and location `1001h` contains `33h`, the instruction POP *HL* results in register pair HL containing `3355h`, and the Stack Pointer containing `1002h`.

POP IX

Operation

IXH ← (SP+1), IXL ← (SP)

Op Code

POP

Operand

IX

1	1	0	1	1	1	0	1	DD
1	1	1	0	0	0	0	1	E1

Description

The top two bytes of the external memory last-in, first-out (LIFO) stack are popped to Index Register IX. The Stack Pointer (SP) Register pair holds the 16-bit address of the current top of the Stack. This instruction first loads to the low-order portion of IX the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded to the high-order portion of IX. The SP is incremented again.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected

None.

Example

If the Stack Pointer contains `1000h`, memory location `1000h` contains `55h`, and location `1001h` contains `33h`, the instruction `POP IX` results in Index Register IX containing `3355h`, and the Stack Pointer containing `1002h`.

POP IY

Operation

IYH ← (SP – X1), IYL ← (SP)

Op Code

POP

Operand

IY

1	1	0	1	1	1	0	1	DD
1	1	1	1	1	1	0	1	FD

Description

The top two bytes of the external memory last-in, first-out (LIFO) stack are popped to Index Register IY. The Stack Pointer (SP) Register pair holds the 16-bit address of the current top of the Stack. This instruction first loads to the low-order portion of IY the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded to the high-order portion of IY. The SP is incremented again.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected

None.

Example

If the Stack Pointer Contains `1000h`, memory location `1000h` contains `55h`, and location `1001h` contains `33h`, the instruction POP *IY* results in Index Register IY containing `3355h`, and the Stack Pointer containing `1002h`.

Exchange, Block Transfer, and Search Group

The following exchange, block transfer, and search group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

EX DE, HL

Operation

DE ↔ HL

Op Code

EX

Operands

DE, HL

1	1	1	0	1	0	1	1	EB

Description

The 2-byte contents of register pairs DE and HL are exchanged.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

Example

If register pair DE contains `2822h` and register pair HL contains `499Ah`, then upon the execution of an EX *DE, HL* instruction, register pair DE contains `499Ah` and register pair HL contains `2822h`.

EX AF, AF'

Operation

AF ↔ AF'

Op Code

EX

Operands

AF, AF′

0	0	0	0	1	0	0	0	08

Description

The 2-byte contents of the register pairs AF and AF' are exchanged. Register pair AF consists of registers A′ and F′.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

Example

If register pair AF contains `9900h` and register pair AF′ contains `5944h`, the contents of AF are `5944h` and the contents of AF′ are `9900h` upon execution of the EX *AF, AF′* instruction.

EXX

Operation

(BC) ↔ (BC′), (DE) ↔ (DE'), (HL) ↔ (HL′)

Op Code

EXX

Operands

None.

1	1	0	1	1	0	0	1	D9

Description

Each 2-byte value in register pairs BC, DE, and HL is exchanged with the 2-byte value in BC', DE', and HL', respectively.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

Example

If register pairs BC, DE, and HL contain `445Ah`, `3DA2h`, and `8859h`, respectively, and register pairs BC', DE', and HL' contain `0988h`, `9300h`, and `00E7h`, respectively, then upon the execution of an EXX instruction, BC' contains `0988h`; DE' contains `9300h`; HL contains `00E7h`; BC' contains `445Ah`; DE' contains `3DA2h`; and HL' contains `8859h`.

EX (SP), HL

Operation

H ↔ (SP+1), L ↔ (SP)

Op Code

EX

Operands

(SP), HL

1	1	1	0	0	0	1	1	E3

Description

The low-order byte contained in register pair HL is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high-order byte of HL is exchanged with the next highest memory address (SP+1).

M Cycles	T States	4 MHz E.T.
5	19 (4, 3, 4, 3, 5)	4.75

Condition Bits Affected

None.

Example

If the HL register pair contains `7012h`, the SP register pair contains `8856h`, the memory location `8856h` contains byte `11h`, and memory location `8857h` contains byte `22h`, then the instruction EX *(SP), HL* results in the HL register pair containing number `2211h`, memory location `8856h` containing byte `12h`, memory location `8857h` containing byte `70h` and Stack Pointer containing `8856h`.

EX (SP), IX

Operation

IXH ↔ (SP+1), IXL ↔ (SP)

Op Code

EX

Operands

(SP), IX

1	1	0	1	1	1	0	1	DD
1	1	1	0	0	0	1	1	E3

Description

The low-order byte in Index Register IX is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high-order byte of IX is exchanged with the next highest memory address (SP+1).

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 4, 3, 5)	5.75

Condition Bits Affected

None.

Example

If Index Register IX contains `3988h`, the SP register pair Contains `0100h`, memory location `0100h` contains byte `90h`, and memory location `0101h` contains byte `48h`, then the instruction EX *(SP)*, *IX* results in the IX register pair containing number `4890h`, memory location `0100h` containing `88h`, memory location `0101h` containing `39h`, and the Stack Pointer containing `0100h`.

EX (SP), IY

Operation

IYH ↔ (SP+1), IYL ↔ (SP)

Op Code

EX

Operands

(SP), IY

1	1	1	1	1	1	0	1	FD
1	1	1	0	0	0	1	1	E3

Description

The low-order byte in Index Register IY is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high-order byte of IY is exchanged with the next highest memory address (SP+1).

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 4, 3, 5)	5.75

Condition Bits Affected

None.

Example

If Index Register IY contains `3988h`, the SP register pair contains `0100h`, memory location `0100h` contains byte `90h`, and memory location `0101h` contains byte `48h`, then the instruction EX *(SP)*, *IY* results in the IY register pair containing number `4890h`, memory location `0100h` containing `88h`, memory location `0101h` containing `39h`, and the Stack Pointer containing `0100h`.

LDI

Operation

(DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC ← BC – 1

Op Code

LDI

Operands

(SP), HL

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	0	0	A0

Description

A byte of data is transferred from the memory location addressed, by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both these register pairs are incremented and the Byte Counter (BC) Register pair is decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is set if BC – 1 ≠ 0; otherwise, it is reset.

N is reset.

C is not affected.

Example

If the HL register pair contains `1111h`, memory location `1111h` contains byte `88h`, the DE register pair contains `2222h`, the memory location `2222h` contains byte `66h`, and the BC register pair contains 7h, then the instruction LDI results in the following contents in register pairs and memory addresses:

HL	contains	1112h
(1111h)	contains	88h
DE	contains	2223h
(2222h)	contains	88h
BC	contains	6H

LDIR

Operation

(DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC F ↔ BC – 1

Op Code

LDIR

Operand

B8

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	0	0	B0

Description

This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the DE register pair. Both these register pairs are incremented and the Byte Counter (BC) Register pair is decremented. If decrementing allows the BC to go to 0, the instruction is terminated. If BC is not 0, the program counter is decremented by two and the instruction is repeated. Interrupts are recognized and two refresh cycles are executed after each data transfer. When the BC is set to 0 prior to instruction execution, the instruction loops through 64KB.

For BC ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

For BC = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is reset.

N is reset.

C is not affected.

Example

The HL register pair contains `11111h`, the DE register pair contains `2222h`, the BC register pair contains `0003h`, and memory locations contain the following data.

(1111h)	contains	88h	(2222h)	contains	66h
(1112h)	contains	36h	(2223h)	contains	59h
(1113h)	contains	A5h	(2224h)	contains	C5h

Upon the execution of an LDIR instruction, the contents of register pairs and memory locations now contain:

HL	contains	1114h			
DE	contains	2225h			
BC	contains	0000h			
(1111h)	contains	88h	(2222h)	contains	88h
(1112h)	contains	36h	(2223h)	contains	36h
(1113h)	contains	A5h	(2224h)	contains	A5h

LDD

Operation

(DE) ← (HL), DE ← DE – 1, HL ← HL– 1, BC ← BC– 1

Op Code

LDD

Operands

None.

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	0	0	A8

Description

This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these register pairs including the Byte Counter (BC) Register pair are decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is set if BC – 1 ≠ 0; otherwise, it is reset.

N is reset.

C is not affected.

Example

If the HL register pair contains `1111h`, memory location `1111h` contains byte `88h`, the DE register pair contains `2222h`, memory location `2222h` contains byte `66h`, and the BC register pair contains 7h, then instruction LDD results in the following contents in register pairs and memory addresses:

HL	contains	1110h
(1111h)	contains	88h
DE	contains	2221h
(2222h)	contains	88h
BC	contains	6h

LDDR

Operation

(DE) ← (HL), DE ← D ← 1, HL ← HL– 1, BC ← BC – 1

Op Code

LDDR

Operands

None.

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	0	0	B8

Description

This 2-byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these registers, and the BC (Byte Counter), are decremented. If decrementing causes BC to go to 0, the instruction is terminated. If BC is not 0, the program counter is decremented by two and the instruction is repeated. Interrupts are recognized and two refresh cycles execute after each data transfer.

When the BC is set to 0, prior to instruction execution, the instruction loops through 64KB.

For BC ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

For BC = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is reset.

N is reset.

Example

The HL register pair contains `1114h`, the DE register pair contains `2225h`, the BC register pair contains `0003h`, and memory locations contain the following data.

(1114h)	contains	A5h	(2225h)	contains	C5h
(1113h)	contains	36h	(2224h)	contains	59h
(1112h)	contains	88h	(2223h)	contains	66h

Upon the execution of an LDDR instruction, the contents of the register pairs and memory locations now contain:

HL	contains	1111h			
DE	contains	2222h			
DC	contains	0000h			
(1114h)	contains	A5h	(2225h)	contains	A5h
(1113h)	contains	36h	(2224h)	contains	36h
(1112h)	contains	88h	(2223h)	contains	88h

CPI

Operation

A – (HL), HL ← HL +1, BC ← BC – 1

Op Code

CPI

Operands

None.

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	0	0	A1

Description

The contents of the memory location addressed by the HL register is compared with the contents of the Accumulator. With a true compare, a condition bit is set. Then HL is incremented and the Byte Counter (register pair BC) is decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if A is (HL); otherwise, it is reset.

H is set if borrow from bit 4; otherwise, it is reset.

P/V is set if BC – 1 is not 0; otherwise, it is reset.

N is set.

C is not affected.

Example

If the HL register pair contains `1111h`, memory location `1111h` contains `3Bh`, the Accumulator contains `3Bh`, and the Byte Counter contains `0001h`. Upon the execution of a CPI instruction, the Byte Counter contains `0000h`, the HL register pair contains `1112h`, the Z flag in the F register is set, and the P/V flag in the F Register is reset. There is no effect on the contents of the Accumulator or to address `1111h`.

CPIR

Operation

A – (HL), HL ← HL+1, BC ← BC – 1

Op Code

CPIR

Operands

None.

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	0	1	B1

Description

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. During a compare operation, a condition bit is set. HL is incremented and the Byte Counter (register pair BC) is decremented. If decrementing causes BC to go to 0 or if A = (HL), the instruction is terminated. If BC is not 0 and A ≠ (HL), the program counter is decremented by two and the instruction is repeated. Interrupts are recognized and two refresh cycles are executed after each data transfer.

If BC is set to 0 before instruction execution, the instruction loops through 64KB if no match is found.

For BC ≠ 0 and A ≠ (HL):

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

For BC = 0 and A = (HL):

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if A equals (HL); otherwise, it is reset.

H is set if borrow from bit 4; otherwise, it is reset.

P/V is set if BC – 1 does not equal 0; otherwise, it is reset.

N is set.

C is not affected.

Example

If the HL register pair contains `1111h`, the Accumulator contains `F3h`, the Byte Counter contains `0007h`, and memory locations contain the following data.

(1111h)	contains	52h
(1112h)	contains	00h
(1113h)	contains	F3h

Upon the execution of a CPIR instruction, register pair HL contains `1114h`, the Byte Counter contains `0004h`, the P/V flag in the F Register is set, and the Z flag in the F Register is set.

CPD

Operation

A – (HL), HL ← HL – 1, BC ← BC – 1

Op Code

CPD

Operands

None.

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	0	1	A9

Description

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. During a compare operation, a condition bit is set. The HL and Byte Counter (register pair BC) are decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if A equals (HL); otherwise, it is reset.

H is set if borrow from bit 4; otherwise, it is reset.

P/V is set if BC – 1 x 0; otherwise, it is reset.

N is set.

C is not affected.

Example

If the HL register pair contains `1111h`, memory location `1111h` contains `3Bh`, the Accumulator contains `3Bh`, and the Byte Counter contains `0001h`. Upon the execution of a CPD instruction, the Byte Counter contains `0000h`, the HL register pair contains `1110h`, the flag in the F Register is set, and the P/V flag in the F Register is reset. There is no effect on the contents of the Accumulator or address `1111h`.

CPDR

Operation

A – (HL), HL ← HL – 1, BC ← BC – 1

Op Code

CPDR

Operands

None.

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	0	1	B9

Description

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. During a compare operation, a condition bit is set. The HL and Byte Counter (BC) Register pairs are decremented. If decrementing allows the BC to go to 0 or if A = (HL), the instruction is terminated. If BC is not 0 and A = (HL), the program counter is decremented by two and the instruction is repeated. Interrupts are recognized and two refresh cycles execute after each data transfer. When the BC is set to 0, prior to instruction execution, the instruction loops through 64KB if no match is found.

For BC ≠ 0 and A ≠ (HL):

M Cycles	T States	4 MHz E.T.
5	21 (4, 4, 3, 5, 5)	5.25

For BC = 0 and A = (HL):

M Cycles	T States	4 MHz E.T.
4	16 (4, 4, 3, 5)	4.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if A = (HL); otherwise, it is reset.

H is set if borrow form bit 4; otherwise, it is reset.

P/V is set if BC – 1 ≠ 0; otherwise, it is reset.

N is set.

C is not affected.

Example

The HL register pair contains `1118h`, the Accumulator contains `F3h`, the Byte Counter contains `0007h`, and memory locations contain the following data.

(1118h)	contains	52h
(1117h)	contains	00h
(1116h)	contains	F3h

Upon the execution of a CPDR instruction, register pair HL contains `1115h`, the Byte Counter contains `0004h`, the P/V flag in the F Register is set, and the Z flag in the F Register is set.

8-Bit Arithmetic Group

The following 8-bit arithmetic group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

ADD A, r

Operation

A ← A + r

Op Code

ADD

Operands

A, r

1	0	0	0	0	◄──	r	──►

Description

The contents of register *r* are added to the contents of the Accumulator, and the result is stored in the Accumulator. The *r* symbol identifies the registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is set if carry from bit 7; otherwise, it is reset.

Example

If the Accumulator contains `44h` and Register C contains `11h`, then upon the execution of an ADD *A, C* instruction, the Accumulator contains `55h`.

ADD A, n

Operation

A ← A + n

Op Code

ADD

Operands

A, *n*

1	1	0	0	0	1	1	0	C6
◄			n				►	

Description

The *n* integer is added to the contents of the Accumulator, and the results are stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is set if carry from bit 7; otherwise, it is reset.

Example

If the Accumulator contains 23h, then upon the execution of an ADD *A*, 33h instruction, the Accumulator contains 56h.

ADD A, (HL)

Operation

A ← A + (HL)

Op Code

ADD

Operands

A, (HL)

1	0	0	0	0	1	1	0	86

Description

The byte at the memory address specified by the contents of the HL register pair is added to the contents of the Accumulator, and the result is stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is set if carry from bit 7; otherwise, it is reset.

Example

If the Accumulator contains `A0h`, register pair HL contains `2323h`, and memory location `2323h` contains byte `08h`, then upon the execution of an ADD *A, (HL)* instruction, the Accumulator contains `A8h`.

ADD A, (IX + d)

Operation

A ← A + (IX+d)

Op Code

ADD

Operands

A, (IX + d)

1	1	0	1	1	1	0	1	DD
1	0	0	0	0	1	1	0	86
◄			d				►	

Description

The contents of the Index (register pair IX) Register is added to a two's complement displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is set if carry from bit 7; otherwise, it is reset.

Example

If the Accumulator contains `11h`, Index Register IX contains `1000h`, and memory location `1005h` contains `22h`, then upon the execution of an ADD *A, (IX* + `5h`) instruction, the Accumulator contains `33h`.

ADD A, (IY + d)

Operation

A ← A + (IY+d)

Op Code

ADD

Operands

A, (IY + d)

1	1	1	1	1	1	0	1	FD
1	0	0	0	0	1	1	0	86
◄			d				►	

Description

The contents of the Index (register pair IY) Register is added to a two's complement displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator, and the result is stored in the Accumulator.

M Cycles	T States	4 MHz E.T.
5	19(4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3: otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is set if carry from bit 7; otherwise, it is reset.

Example

If the Accumulator contains `11h`, Index Register IY contains `1000h`, and memory location `1005h` contains `22h`, then upon the execution of an ADD *A*, (*IY* + `5h`) instruction, the Accumulator contains `33h`.

ADC A, s

Operation

A ← A + s + CY

Op Code

ADC

Operands

A, s

This *s* operand is any of *r*, *n*, *(HL)*, *(IX+d)*, or *(lY+d)* as defined for the analogous ADD instruction. These possible op code/operand combinations are assembled as follows in the object code:

ADC A,r	1	0	0	0	1	◄—	r*	—►	
ADC A,n	1	1	0	0	1	1	1	0	CE
	◄—			n				—►	
ADC A, (HL)	1	0	0	0	1	1	1	0	8E
ADC A, (IX+d)	1	1	0	1	1	1	1	0	DD
	1	0	0	0	1	1	1	0	8E
	◄—			d				—►	
ADC A, (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	0	0	1	1	1	0	8E
	◄—			d				—►	

r identifies registers B, C, D, E, H, L, or A, assembled as follows in the object code field:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

The *s* operand, along with the Carry Flag (C in the F Register) is added to the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycle	T States	4 MHz E.T.
ADC A, r	1	4	1.00
ADC A, n	2	7 (4, 3)	1.75
ADC A, (HL)	2	7 (4, 3)	1.75
ADC A, (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
ADC A, (lY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is set if carry from bit 7: otherwise, it is reset.

Example

If the Accumulator contents are `16h`, the Carry Flag is set, the HL register pair contains `6666h`, and address `6666h` contains `10h`, then upon the execution of an ADC *A, (HL)* instruction, the Accumulator contains `27h`.

SUB s

Operation

A ← A – s

Op Code

SUB

Operand

s

This *s* operand is any of *r*, *n*, *(HL)*, *(IX+d)*, or *(lY+d)* as defined for the analogous ADD instruction. These possible op code/operand combinations are assembled as follows in the object code:

SUB r	1	0	0	1	0	◄	r*	►	
SUB n	1	1	0	1	0	1	1	0	D6
	◄			n				►	
SUB (HL)	1	0	0	1	0	1	1	0	96
SUB (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	0	1	0	1	1	0	96
	◄			d				►	
SUB (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	0	1	0	1	1	0	96
	◄			d				►	

r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

The *s* operand is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycle	T States	4 MHz E.T.
SUB r	1	4	1.00
SUB n	2	7 (4, 3)	1.75
SUB (HL)	2	7 (4, 3)	1.75
SUB (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
SUB (lY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if borrow from bit 4; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is set.

C is set if borrow; otherwise, it is reset.

Example

If the Accumulator contents are `29h`, and the D Register contains `11h`, then upon the execution of a SUB *D* instruction, the Accumulator contains `18h`.

SBC A, s

Operation

A ← A – s – CY

Op Code

SBC

Operands

A, s

The *s* operand is any of *r*, *n*, *(HL)*, *(IX+d)*, or *(lY+d)* as defined for the analogous ADD instructions. These possible op code/operand combinations are assembled as follows in the object code.

SBC A, r	1	0	0	1	1	←	r*	→	
SBC A, n	1	1	0	1	1	1	1	0	DE
	←			n				→	
SBC A, (HL)	1	0	0	1	1	1	1	0	9E
SBC A, (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	0	1	1	1	1	0	9E
	←			d				→	
SBC A, (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	0	1	1	1	1	0	9E
	←			d				→	

r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description

The *s* operand, along with the Carry flag (C in the F Register) is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
SBC A, r	1	4	1.00
SBC A, n	2	7(4, 3)	1.75
SBC A, (HL)	2	7 (4, 3)	1.75
SBC A, (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
SBC A, (lY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if borrow from bit 4; otherwise, it is reset.

P/V is reset if overflow; otherwise, it is reset.

N is set.

C is set if borrow; otherwise, it is reset.

Example

If the Accumulator contains `16h`, the carry flag is set, the HL register pair contains `3433h`, and address `3433h` contains `05h`, then upon the execution of an SBC *A, (HL)* instruction, the Accumulator contains `10h`.

AND s

Operation

A ← A ∧ s

Op Code

AND

Operand

s

The *s* operand is any of *r*, *n*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous ADD instructions. These possible op code/operand combinations are assembled as follows in the object code:

AND r*	1	0	1	0	0	←	r*	→	
AND n	1	1	1	0	0	1	1	0	E6
	←			n				→	
AND (HL)	1	0	1	0	0	1	1	0	A6
AND (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	0	0	1	1	0	A6
	←			d				→	
AND (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	0	0	1	1	0	A6
	←			d				→	

r identifies registers B, C, D, E, H, L, or A specified in the assembled object code field, as follows:

Register	**r**
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

A logical AND operation is performed between the byte specified by the *s* operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	**M Cycles**	**T States**	**4 MHz E.T.**
AND r	1	4	1.00
AND n	2	7 (4, 3)	1.75
AND (HL)	2	7 (4, 3)	1.75
AND (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
AND (IX+d)	5	19 (4, 4, 3. 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set.

P/V is reset if overflow; otherwise, it is reset.

N is reset.

C is reset.

Example

If Register B contains `7Bh` (`0111 1011`) and the Accumulator contains `C3h` (`1100 0011`), then upon the execution of an AND *B* instruction, the Accumulator contains `43h` (`0100 0011`).

OR s

Operation

A ← A ∨ s

Op Code

OR

Operand

s

The *s* operand is any of *r*, *n*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous ADD instructions. These possible op code/operand combinations are assembled as follows in the object code:

OR r*	1	0	1	1	0	◄	r*	►	
OR n	1	1	1	1	0	1	1	0	F6
	◄			n				►	
OR (HL)	1	0	1	1	0	1	1	0	B6
OR (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	1	0	1	1	0	B6
	◄			d				►	
OR (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	1	0	1	1	0	B6
	◄			d				►	

r identifies registers B, C–, D, E, H, L, or A specified in the assembled object code field, as follows:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

A logical OR operation is performed between the byte specified by the *s* operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
OR r	1	4	1.00
OR n	2	7 (4, 3)	1.75
OR (HL)	2	7 (4, 3)	1.75
OR (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
OR (lY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is reset.

Example

If the H Register contains `48h` (`0100 0100`), and the Accumulator contains `12h` (`0001 0010`), then upon the execution of an OR *H* instruction, the Accumulator contains `5Ah` (`0101 1010`).

XOR s

Operation

A ← A ⊕ s

Op Code

XOR

Operand

s

The *s* operand is any of *r*, *n*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous ADD instructions. These possible Op Code/operand combinations are assembled as follows in the object code:

OR r*	1	0	1	1	0	◄—	r*	—►	
OR n	1	1	1	1	0	1	1	0	F6
	◄—				n			—►	
OR (HL)	1	0	1	1	0	1	1	0	B6
OR (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	1	0	1	1	0	B6
	◄—				d			—►	
OR (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	1	0	1	1	0	B6
	◄—				d			—►	

r identifies registers B, C, D, E, H, L, or A specified in the assembled object code field, as follows:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`1l1`

Description

The logical exclusive-OR operation is performed between the byte specified by the *s* operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
XOR r	1	4	1.00
XOR n	2	7 (4, 3)	1.75
XOR (HL)	2	7 (4, 3)	1.75
XOR (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
XOR (lY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is reset.

Example

If the Accumulator contains `96h` (`1001 0110`), then upon the execution of an XOR `5Dh` (`5Dh` = `0101 1101`) instruction, the Accumulator contains `CBh` (`1100 1011`).

CP s

Operation

A – s

Op Code

CP

Operand

s

The *s* operand is any of *r*, *n*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous ADD instructions. These possible op code/operand combinations are assembled as follows in the object code:

CP r*	1	0	1	1	1	◄	r*	►	
CP n	1	1	1	1	1	1	1	0	FE
	◄				n			►	
CP (HL)	1	0	1	1	1	1	1	0	BE
CP (IX+d)	1	1	0	1	1	1	0	1	DD
	1	0	1	1	1	1	1	0	BE
	◄				d			►	
CP (IY+d)	1	1	1	1	1	1	0	1	FD
	1	0	1	1	1	1	1	0	BE
	◄				d			►	

r identifies registers B, C, D, E, H, L, or A specified in the assembled object code field, as follows:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

The contents of the *s* operand are compared with the contents of the Accumulator. If there is a true compare, the Z flag is set. The execution of this instruction does not affect the contents of the Accumulator.

Instruction	M Cycles	T States	4 MHz E.T.
CP r	1	4	1.00
CP n	2	7(4, 3)	1.75
CP (HL)	2	7 (4, 3)	1.75
CP (IX+d)	5	19 (4, 4, 3, 5, 3)	4.75
CP (lY+d)	5	19 (4, 4, 3, 5, 3)	4.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if borrow from bit 4; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is set.

C is set if borrow; otherwise, it is reset.

Example

If the Accumulator contains `63h`, the HL register pair contains `6000h`, and memory location `6000h` contains `60h`, the instruction CP *(HL)* results in the PN flag in the F Register resetting.

INC r

Operation

r ← r + 1

Op Code

INC

Operand

r

0	0	◄—	r	—►	1	0	0

Description

Register *r* is incremented and register *r* identifies any of the registers A, B, C, D, E, H, or L, assembled as follows in the object code.

Register	r
A	`111`
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if *r* was `7Fh` before operation; otherwise, it is reset.

N is reset.

C is not affected.

Example

If the D Register contains `28h`, then upon the execution of an INC *D* instruction, the D Register contains `29h`.

INC (HL)

Operation

(HL) ← (HL) + 1

Op Code

INC

Operand

(HL)

0	0	1	1	0	1	0	0	34

Description

The byte contained in the address specified by the contents of the HL register pair is incremented.

M Cycles	T States	4 MHz E.T.
3	11 (4, 4, 3)	2.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if (HL) was `7Fh` before operation; otherwise, it is reset.

N is reset.

C is not affected.

Example

If the HL register pair contains `3434h` and address `3434h` contains `82h`, then upon the execution of an INC *(HL)* instruction, memory location `3434h` contains `83h`.

INC (IX+d)

Operation

(IX+d) ← (IX+d) + 1

Op Code

INC

Operands

(IX+d)

1	1	0	1	1	1	0	1	DD
0	0	1	1	0	1	0	0	34
◄				d			►	

Description

The contents of Index Register IX (register pair IX) are added to the two's-complement displacement integer, *d*, to point to an address in memory. The contents of this address are then incremented.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if *(IX+d)* was `7Fh` before operation; otherwise, it is reset.

N is reset.

C is not affected.

Example

If Index Register pair IX contains `2020h` and memory location `2030h` contains byte `34h`, then upon the execution of an INC *(IX+*`10h`*)* instruction, memory location `2030h` contains `35h`.

INC (IY+d)

Operation

(lY+d) ← (lY+d) + 1

Op Code

INC

Operands

(lY+d)

1	1	1	1	1	1	0	1	FD
0	0	1	1	0	1	0	0	34
◄			d				►	

Description

The contents of Index Register IY (register pair IY) are added to the two's-complement displacement integer, *d*, to point to an address in memory. The contents of this address are then incremented.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 3; otherwise, it is reset.

P/V is set if *(lY+d)* was `7Fh` before operation; otherwise, it is reset.

N is reset.

C is not affected.

Example

If Index Register IY are `2020h` and memory location `2030h` contains byte `34h`, then upon the execution of an INC *(IY+*`10h`*)* instruction, memory location `2030h` contains `35h`.

DEC m

Operation

m ← m – 1

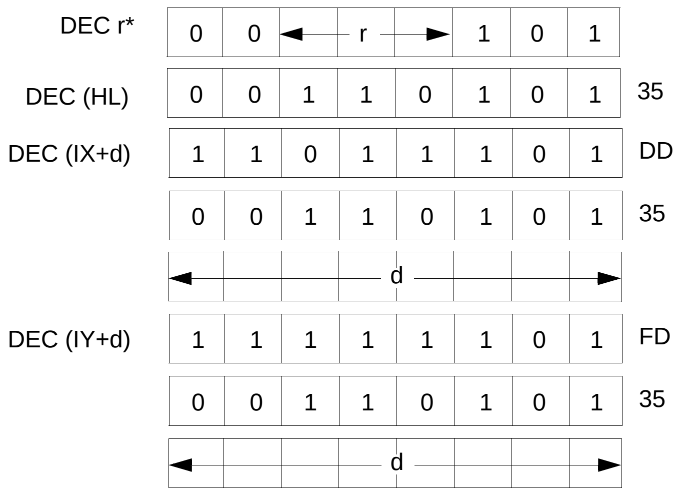
Op Code

DEC

Operand

m

The *m* operand is any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous INC instructions. These possible op code/operand combinations are assembled as follows in the object code:



r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	**r**
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description

The byte specified by the *m* operand is decremented.

Instruction	M Cycles	T States	4 MHz E.T.
DEC r	1	4	1.00
DEC (HL)	3	11 (4, 4, 3)	2.75
DEC (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
DEC (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if borrow from bit 4, otherwise, it is reset.

P/V is set if *m* was `80h` before operation; otherwise, it is reset.

N is set.

C is not affected.

Example

If the D Register contains byte `2Ah`, then upon the execution of a DEC *D* instruction, the D Register contains `29h`.

General-Purpose Arithmetic and CPU Control Groups

The following general-purpose arithmetic and CPU control group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

DAA

Operation

@

Op Code

DAA

0	0	1	0	0	1	1	1	27

Operands

None.

Description

This instruction conditionally adjusts the Accumulator for BCD addition and subtraction operations. For addition (ADD, ADC, INC) or subtraction (SUB, SBC, DEC, NEG), the following table indicates the operation being performed:

Operation	C Before DAA	Hex Value In Upper Digit (Bits 7–4)	H Before DAA	Hex Value In Lower Digit (Bits 3–0)	Number Added To Byte	C After DAA
	0	9–0	0	0–9	00	0
	0	0–8	0	A–F	06	0
	0	0–9	1	0–3	06	0
ADD	0	A–F	0	0–9	60	1
ADC	0	9–F	0	A–F	66	1
INC	0	A–F	1	0–3	66	1
	1	0–2	0	0–9	60	1
	1	0–2	0	A–F	66	1
	1	0–3	1	0–3	66	1
SUB	0	0–9	0	0–9	00	0
SBC	0	0–8	1	6–F	FA	0
DEC	1	7–F	0	0–9	A0	1
NEG	1	6–7	1	6–F	9A	1

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is set if most-significant bit of the Accumulator is 1 after an operation; otherwise, it is reset.

Z is set if the Accumulator is 0 after an operation; otherwise, it is reset.

H: see the DAA instruction table on the previous page.

P/V is set if the Accumulator is at even parity after an operation; otherwise, it is reset.

N is not affected.

C: see the DAA instruction table on the previous page.

Example

An addition operation is performed between 15 (BCD) and 27 (BCD); simple decimal arithmetic provides the following result:

$$\begin{array}{r} 15 \\ \underline{+\ 27} \\ 42 \end{array}$$

The binary representations are added in the Accumulator according to standard binary arithmetic, as follows:

$$\begin{array}{rrl} 0001 & 0101 & \\ \underline{+\ 0010} & \underline{0111} & \\ 0011 & 1100 & = 3C \end{array}$$

The sum is ambiguous. The DAA instruction adjusts this result so that the correct BCD representation is obtained, as follows:

$$\begin{array}{rrl} 0011 & 1100 & \\ \underline{+\ 0000} & \underline{0110} & \\ 0100 & 0010 & = 42 \end{array}$$

CPL

Operation

$A \leftarrow \overline{A}$

Op Code

CPL

0	0	1	0	1	1	1	1	2F

Operands

None.

Description

The contents of the Accumulator (Register A) are inverted (one's complement).

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is set.

P/V is not affected.

N is set.

C is not affected.

Example

If the Accumulator contains `1011 0100`, then upon the execution of a CPL instruction, the Accumulator contains `0100 1011`.

NEG

Operation

A ← 0 – A

Op Code

NEG

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	0	44

Operands

None.

Description

The contents of the Accumulator are negated (two's complement). This method is the same as subtracting the contents of the Accumulator from zero.

Note: The `80h` address remains unchanged.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if borrow from bit 4; otherwise, it is reset.

P/V is set if Accumulator was `80h` before operation; otherwise, it is reset.

N is set.

C is set if Accumulator was not `00h` before operation; otherwise, it is reset.

Example

The Accumulator contains the following data:

1	0	0	1	1	0	0	0

Upon the execution of a NEG instruction, the Accumulator contains:

0	1	1	0	1	0	0	0

CCF

Operation

$CY \leftarrow \overline{CY}$

Op Code

CCF

0	0	1	1	1	1	1	1	3F

Operands

None.

Description

The Carry flag in the F Register is inverted.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is not affected.

Z is not affected.

H, previous carry is copied.

P/V is not affected.

N is reset.

C is set if CY was 0 before operation; otherwise, it is reset.

SCF

Operation

CY ← 1

Op Code

SCF

0	0	1	1	0	1	1	1	37

Operands

None.

Description

The Carry flag in the F Register is set.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is not affected.

N is reset.

C is set.

NOP

Operation

—

Op Code

NOP

0	0	0	0	0	0	0	0	00

Operands

None.

Description

The CPU performs no operation during this machine cycle.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

HALT

Operation

—

Op Code

HALT

0	1	1	1	0	1	1	0	76

Operands

None.

Description

The HALT instruction suspends CPU operation until a subsequent interrupt or reset is received. While in the HALT state, the processor executes NOPs to maintain memory refresh logic.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

DI

Operation

IFF ← 0

Op Code

DI

1	1	1	1	0	0	1	1	F3

Operands

None.

Description

DI disables the maskable interrupt by resetting the interrupt enable flip-flops (IFF1 and IFF2).

Note: This instruction disables the maskable interrupt during its execution.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

Example

When the CPU executes the instruction `DI` the maskable interrupt is disabled until it is subsequently re-enabled by an EI instruction. The CPU does not respond to an Interrupt Request (INT) signal.

EI

Operation

IFF ← 1

Op Code

EI

1	1	1	1	1	0	1	1	FB

Operands

None.

Description

The enable interrupt instruction sets both interrupt enable flip flops (IFFI and IFF2) to a logic 1, allowing recognition of any maskable interrupt.

Note: During the execution of this instruction and the following instruction, maskable interrupts are disabled.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

Example

When the CPU executes an EI RETI instruction, the maskable interrupt is enabled then upon the execution of an the RETI instruction.

IM 0

Operation

Set Interrupt Mode 0

Op Code

IM

Operand

0

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	1	0	46

Description

The IM 0 instruction sets Interrupt Mode 0. In this mode, the interrupting device can insert any instruction on the data bus for execution by the CPU. The first byte of a multi-byte instruction is read during the interrupt acknowledge cycle. Subsequent bytes are read in by a normal memory read sequence.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected

None.

IM 1

Operation

Set Interrupt Mode 1

Op Code

IM

Operand

1

1	1	1	0	1	1	0	1	ED
0	1	0	1	0	1	1	0	56

Description

The IM 1 instruction sets Interrupt Mode 1. In this mode, the processor responds to an interrupt by executing a restart at address `0038h`.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected

None.

IM 2

Operation

Set Interrupt Mode 2

Op Code

IM

Operand

2

1	1	1	0	1	1	0	1	ED
0	1	0	1	1	1	1	0	5E

Description

The IM 2 instruction sets the vectored Interrupt Mode 2. This mode allows an indirect call to any memory location by an 8-bit vector supplied from the peripheral device. This vector then becomes the least-significant eight bits of the indirect pointer, while the I Register in the CPU provides the most-significant eight bits. This address points to an address in a vector table that is the starting address for the interrupt service routine.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected

None.

16-Bit Arithmetic Group

The following 16-bit arithmetic group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

ADD HL, ss

Operation

HL ← HL + ss

Op Code

ADD

Operands

HL, ss

0	0	s	s	1	0	0	1

Description

The contents of register pair ss (any of register pairs BC, DE, HL, or SP) are added to the contents of register pair HL and the result is stored in HL. In the assembled object code, operand *ss* is specified as follows:

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
3	11 (4, 4, 3)	2.75

Condition Bits Affected

S is not affected.

Z is not affected.

H is set if carry from bit 11; otherwise, it is reset.

P/V is not affected.

N is reset.

C is set if carry from bit 15; otherwise, it is reset.

Example

If register pair HL contains the integer `4242h` and register pair DE contains `1111h`, then upon the execution of an ADD *HL*, *DE* instruction, the HL register pair contains `5353h`.

ADC HL, ss

Operation

HL ← HL + ss + CY

Op Code

ADC

Operands

HL, ss

1	1	1	0	1	1	0	1	ED
0	1	s	s	1	0	1	0	

Description

The contents of register pair ss (any of register pairs BC, DE, HL, or SP) are added with the Carry flag (C flag in the F Register) to the contents of register pair HL, and the result is stored in HL. In the assembled object code, operand *ss* is specified as follows:

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if carry from bit 11; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is reset.

C is set if carry from bit 15; otherwise, it is reset.

Example

If register pair BC contains `2222h`, register pair HL contains `5437h`, and the Carry Flag is set, then upon the execution of an ADC *HL*, *BC* instruction, HL contains `765Ah`.

SBC HL, ss

Operation

HL ← HI – ss – CY

Op Code

SBC

Operands

HL, ss

1	1	1	0	1	1	0	1	ED
0	1	s	s	0	0	1	0	

Description

The contents of the register pair ss (any of register pairs BC, DE, HL, or SP) and the Carry Flag (C flag in the F Register) are subtracted from the contents of register pair HL, and the result is stored in HL. In the assembled object code, operand *ss* is specified as follows:

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is set if borrow from bit 12; otherwise, it is reset.

P/V is set if overflow; otherwise, it is reset.

N is set.

C is set if borrow; otherwise, it is reset.

Example

If the HL register pair contains `9999h`, register pair DE contains `1111h`, and the Carry flag is set, then upon the execution of an SBC *HL, DE* instruction, HL contains `8887h`.

ADD IX, pp

Operation

IX ← IX + pp

Op Code

ADD

Operands

IX, pp

1	1	0	1	1	1	0	1	DD
0	0	p	p	1	0	0	1	

Description

The contents of register pair pp (any of register pairs BC, DE, IX, or SP) are added to the contents of Index Register IX, and the results are stored in IX. In the assembled object code, operand *pp* is specified as follows:

Register Pair	ss
BC	00
DE	01
IX	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected

S is not affected.

Z is not affected.

H is set if carry from bit 11; otherwise, it is reset.

P/V is not affected.

N is reset.

C is set if carry from bit 15; otherwise, it is reset.

Example

If Index Register IX contains `333h` and register pair BC contains `5555h`, then upon the execution of an ADD *IX*, *BC* instruction, IX contains `8888h`.

ADD IY, rr

Operation

IY ← IY + rr

Op Code

ADD

Operands

IY, rr

1	1	1	1	1	1	0	1	FD
0	0	r	r	1	0	0	1	

Description

The contents of register pair *rr* (any of register pairs BC, DE, IY, or SP) are added to the contents of Index Register IY, and the result is stored in IY. In the assembled object code, the *rr* operand is specified as follows:

Register Pair	ss
BC	00
DE	01
IY	10
SP	11

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected

S is not affected.

Z is not affected.

H is set if carry from bit 11; otherwise, it is reset.

P/V is not affected.

N is reset.

C is set if carry from bit 15; otherwise, it is reset.

Example

If Index Register IY contains `333h` and register pair BC contains `555h`, then upon the execution of an ADD *IY, BC* instruction, IY contains `8888h`.

INC ss

Operation

ss ← ss + 1

Op Code

INC

Operand

ss

0	0	s	s	0	0	1	1

Description

The contents of register pair ss (any of register pairs BC, DE, HL, or SP) are incremented. In the assembled object code, operand *ss* is specified as follows:

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M Cycles	T States	4 MHz E.T.
1	6	1.50

Condition Bits Affected

None.

Example

If the register pair contains `1000h`, then upon the execution of an INC *HL* instruction, HL contains `1001h`.

INC IX

Operation

IX ← IX + 1

Op Code

INC

Operand

IX

1	1	0	1	1	1	0	1	DD
0	0	1	0	0	0	1	1	23

Description

The contents of Index Register IX are incremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected

None.

Example

If Index Register IX contains the integer `3300h`, then upon the execution of an INC *IX* instruction, Index Register IX contains `3301h`.

INC IY

Operation

IY ← IY + 1

Op Code

INC

Operand

IY

1	1	1	1	1	1	0	1	FD
0	0	1	0	0	0	1	1	23

Description

The contents of Index Register IY are incremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected

None.

Example

If the index register contains `2977h`, then upon the execution of an INC *IY* instruction, Index Register IY contains `2978h`.

DEC ss

Operation

ss ← ss – 1

Op Code

DEC

Operand

ss

0	0	s	s	1	0	1	1

Description

The contents of register pair ss (any of the register pairs BC, DE, HL, or SP) are decremented. In the assembled object code, operand *ss* is specified as follows:

Register Pair	ss
BC	`00`
DE	`01`
HL	`10`
SP	`11`

M Cycles	T States	4 MHz E.T.
1	6	1.50

Condition Bits Affected

None.

Example

If register pair HL contains `1001h`, then upon the execution of an DEC *HL* instruction, HL contains `1000h`.

DEC IX

Operation

IX ← IX – 1

Op Code

DEC

Operand

IX

1	1	0	1	1	1	0	1	DD
0	0	1	0	1	0	1	1	2B

Description

The contents of Index Register IX are decremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected

None.

Example

If Index Register IX contains `2006h`, then upon the execution of a DEC *IX* instruction, Index Register IX contains `2005h`.

DEC IY

Operation

IY ← IY– 1

Op Code

DEC

Operand

IY

1	1	1	1	1	1	0	1	FD
0	0	1	0	1	0	1	1	2B

Description

The contents of Index Register IY are decremented.

M Cycles	T States	4 MHz E.T.
2	10 (4, 6)	2.50

Condition Bits Affected

None.

Example

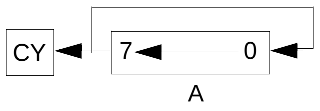
If Index Register IY contains `7649h`, then upon the execution of a DEC *IY* instruction, Index Register IY contains `7648h`.

Rotate and Shift Group

The following rotate and shift group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

RLCA

Operation



Op Code

RLCA

Operands

None.

0	0	0	0	0	1	1	1	07

Description

The contents of the Accumulator (Register A) are rotated left 1 bit position. The sign bit (bit 7) is copied to the Carry flag and also to bit 0. Bit 0 is the least-significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is not affected.

N is reset.

C is data from bit 7 of Accumulator.

Example

The Accumulator contains the following data:

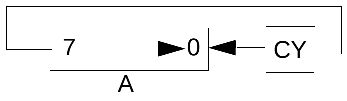
7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

Upon the execution of an RLCA instruction, the Accumulator and Carry flag contains:

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLA

Operation



Op Code

RLA

Operands

None.

0	0	0	1	0	1	1	1	17

Description

The contents of the Accumulator (Register A) are rotated left 1 bit position through the Carry flag. The previous contents of the Carry flag are copied to bit 0. Bit 0 is the least-significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

Condition Bits Affected.

S is not affected.

Z is not affected.

H is reset.

P/V is not affected.

N is reset.

C is data from bit 7 of Accumulator.

Example

The Accumulator and the Carry flag contains the following data:

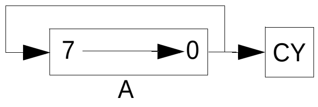
C	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	1	0

Upon the execution of an RLA instruction, the Accumulator and the Carry flag contains:

C	7	6	5	4	3	2	1	0
0	1	1	1	0	1	1	0	1

RRCA

Operation



Op Code

RRCA

Operands

None.

0	0	0	0	1	1	1	1	0F

Description

The contents of the Accumulator (Register A) are rotated right 1 bit position. Bit 0 is copied to the Carry flag and also to bit 7. Bit 0 is the least-significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is not affected.

N is reset.

C is data from bit 0 of Accumulator.

Example

The Accumulator contains the following data.

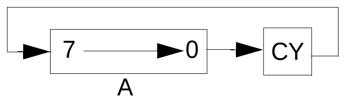
7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	1

Upon the execution of an RRCA instruction, the Accumulator and the Carry flag now contain:

7	6	5	4	3	2	1	0	C
1	0	0	1	1	0	0	0	1

RRA

Operation



Op Code

RRA

Operands

None.

0	0	0	1	1	1	1	1	1F

Description

The contents of the Accumulator (Register A) are rotated right 1 bit position through the Carry flag. The previous contents of the Carry flag are copied to bit 7. Bit 0 is the least-significant bit.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

S is not affected.

Z is not affected.

H is reset.

P/V is not affected.

N is reset.

C is data from bit 0 of Accumulator.

Example

The Accumulator and the Carry Flag contain the following data:

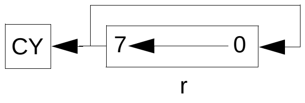
7	6	5	4	3	2	1	0	C
1	1	1	0	0	0	0	1	0

Upon the execution of an RRA instruction, the Accumulator and the Carry flag now contain:

7	6	5	4	3	2	1	0	C
0	1	1	1	0	0	0	0	1

RLC r

Operation



Op Code

RLC

Operand

r

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	←	r	→	

Description

The contents of register *r* are rotated left 1 bit position. The contents of bit 7 are copied to the Carry flag and also to bit 0. In the assembled object code, operand *r* is specified as follows:

Register	**r**
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M Cycles	**T States**	**4 MHz E.T.**
2	8 (4, 4)	2.00

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is data from bit 7 of source register.

Example

Register *r* contains the following data.

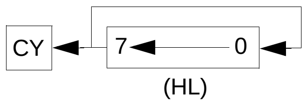
7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

Upon the execution of an RLC *r* instruction, register *r* and the Carry flag now contain:

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLC (HL)

Operation



Op Code

RLC

Operand

(HL)

1	1	0	0	1	0	1	1	CB
0	0	0	0	0	1	1	0	06

Description

The contents of the memory address specified by the contents of register pair HL are rotated left 1 bit position. The contents of bit 7 are copied to the Carry flag and also to bit 0. Bit 0 is the least-significant bit.

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is data from bit 7 of source register.

Example

The HL register pair contains `2828h` and the contents of memory location `2828h` are:

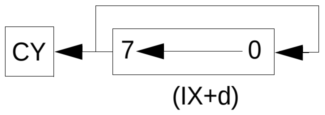
7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

Upon the execution of an RLC*(HL)* instruction, memory location `2828h` and the Carry flag now contain:

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLC (IX+d)

Operation



Op Code

RLC

Operand

(IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
◄			d				►	
0	0	0	0	0	1	1	0	06

Description

The contents of the memory address specified by the sum of the contents of Index Register IX and the two's-complement displacement integer, *d*, are rotated left 1 bit position. The contents of bit 7 are copied to the Carry flag and also to bit 0. Bit 0 is the least-significant bit.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is data from bit 7 of source register.

Example

Index Register IX contains `1000h` and memory location `1022h` contains the following data.

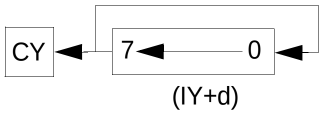
7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

Upon the execution of an RLC *(IX+*`2h`*)* instruction, memory location `1002h` and the Carry flag now contain:

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RLC (IY+d)

Operation



Op Code

RLC

Operand

(lY+d)

1	1	1	1	1	1	0	1	FD
1	1	0	0	1	0	1	1	CB
◄				d			►	
0	0	0	0	0	1	1	0	06

Description

The contents of the memory address specified by the sum of the contents of Index Register IY and the two's-complement displacement integer, *d*, are rotated left 1 bit position. The contents of bit 7 are copied to the Carry flag and also to bit 0. Bit 0 is the least-significant bit.

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is data from bit 7 of source register.

Example

Index Register IY contains `1000h` and memory location `1002h` contain the following data:

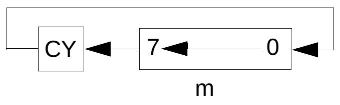
7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0

Upon the execution of an RLC *(IY+*`2h`*)* instruction, memory location `1002h` and the Carry flag now contain:

C	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	1

RL m

Operation



Op Code

PL

Operand

m

The *m* operand is any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous PLC instructions. In the assembled object code, the possible op code/operand combinations are specified as follows:

RL r*	1	1	0	0	1	0	1	1	CB
	0	0	0	1	0	←	r*	→	
RL (HL)	1	1	0	0	1	0	1	1	CB
	0	0	0	1	0	1	1	0	16
RL (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	0	1	0	1	1	0	16
RL (IY+d)	1	1	1	1	1	1	0	1	FB
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	0	1	0	1	1	0	16

r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

The contents of the *m* operand are rotated left 1 bit position. The contents of bit 7 are copied to the Carry flag, and the previous contents of the Carry flag are copied to bit 0.

Instruction	M Cycles	T States	4 MHz E.T.
RL r	2	8 (4, 4)	2.00
RL (HL)	4	15(4, 4, 4, 3)	3.75
RL (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RL (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is data from bit 7 of source register.

Example

The D Register and the Carry flag contain the following data.

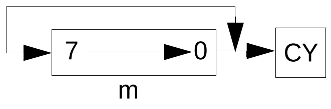
C	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	1	1

Upon the execution of an RL *D* instruction, the D Register and the Carry flag now contain:

C	7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1	0

RRC m

Operation



Op Code

RRC

Operand

m

The *m* operand is any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous RLC instructions. In the assembled object code, the possible op code/operand combinations are specified as follows:

RRC r*	1	1	0	0	1	0	1	1	CB
	0	0	0	0	1	←	r*	→	
RRC (HL)	1	1	0	0	1	0	1	1	CB
	0	0	0	0	1	1	1	0	OE
RRC (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	0	0	1	1	1	0	OE
RRC (IY+d)	1	1	1	1	1	1	0	1	FB
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	0	0	1	1	1	0	OE

r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description

The contents of the *m* operand are rotated right 1 bit position. The contents of bit 0 are copied to the Carry flag and also to bit 7. Bit 0 is the least-significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
RRC r	2	8 (4, 4)	2.00
RRC (HL)	4	15 (4, 4, 4, 3)	3.75
RRC (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RRC (lY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is data from bit 0 of source register.

Example

Register A contains the following data.

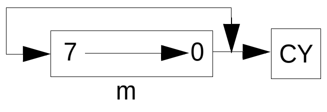
7	6	5	4	3	2	1	0
0	0	1	1	0	0	0	1

Upon the execution of an RRC *A* instruction, Register A and the Carry flag now contain:

7	6	5	4	3	2	1	0	C
1	0	0	1	1	0	0	0	1

RR m

Operation



Op Code

RR

Operand

m

The *m* operand is any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous RLC instructions. In the assembled object code, the possible op code/operand combinations are specified as follows:

RR r*	1	1	0	0	1	0	1	1	CB
	0	0	0	0	1	←	r*	→	
RR (HL)	1	1	0	0	1	0	1	1	CB
	0	0	0	0	1	1	1	0	1E
RR (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	0	1	1	1	1	0	1E
RR (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	0	1	1	1	1	0	1E

r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

The contents of operand *m* are rotated right 1 bit position through the Carry flag. The contents of bit 0 are copied to the Carry flag and the previous contents of the Carry flag are copied to bit 7. Bit 0 is the least-significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
RR r	2	8 (4, 4)	2.00
RR (HL)	4	15 (4, 4, 4, 3)	3.75
RR (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RR (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity even; otherwise, it is reset.

N is reset.

C is data from bit 0 of source register.

Example

The HL register pair contains `4343h` and memory location `4343h` and the Carry flag contain the following data.

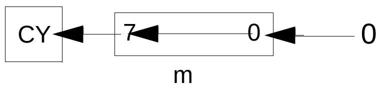
7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	1	0

Upon the execution of an RR *(HL)* instruction, location `4343h` and the Carry flag now contain:

7	6	5	4	3	2	1	0	C
0	1	1	0	1	1	1	0	1

SLA m

Operation



Op Code

SLA

Operand

m

The *m* operand is any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous RLC instructions. In the assembled object code, the possible op code/operand combinations are specified as follows:

SLA r*	1	1	0	0	1	0	1	1	CB
	0	0	1	0	0	←	r*	→	
SLA (HL)	1	1	0	0	1	0	1	1	CB
	0	0	1	0	0	1	1	0	26
SLA (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	1	0	0	1	1	0	26
SLA (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	0	0	1	0	0	1	1	0	26

r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	r
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

An arithmetic shift left 1 bit position is performed on the contents of operand *m*. The contents of bit 7 are copied to the Carry flag. Bit 0 is the least-significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
SLA r	2	8 (4, 4)	2.00
SLA (HL)	4	15 (4, 4, 4, 3)	3.75
SLA (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
SLA (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity is even; otherwise, it is reset.

N is reset.

C is data from bit 7.

Example

Register L contains the following data.

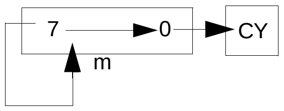
7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1

Upon the execution of an SLA *L* instruction, Register L and the Carry flag now contain:

C	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	1	0

SRA m

Operation



Op Code

SRA

Operand

m

The *m* operand is any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous PLC instructions. In the assembled object code, the possible op code/operand combinations are specified as follows:

SRA r*	1	1	0	0	1	0	1	1	CB
	0	0	1	0	0	◄—	r*	—►	
SRA (HL)	1	1	0	0	1	0	1	1	CB
	0	0	1	0	1	1	1	0	2E
SRA (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	◄—			d				—►	
	0	0	1	0	1	1	1	0	2E
SRA (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	◄—			d				—►	
	0	0	1	0	1	1	1	0	2E

r identifies registers B, C, D, E, H, L, or A assembled as follows in the object code field:

Register	**r**
B	`000`
C	`001`
D	`010`
E	`011`
H	`100`
L	`101`
A	`111`

Description

An arithmetic shift right 1 bit position is performed on the contents of operand *m*. The contents of bit 0 are copied to the Carry flag and the previous contents of bit 7 remain unchanged. Bit 0 is the least-significant bit.

Instruction	**M Cycles**	**T States**	**4 MHz E.T.**
SRA r	2	8 (4, 4)	2.00
SRA (HL)	4	15 (4, 4, 4, 3)	3.75
SRA (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
SRA (IY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is set if result is negative; otherwise, it is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity is even; otherwise, it is reset.

N is reset.

C is data from bit 0 of source register.

Example

Index Register IX contains `1000h` and memory location `1003h` contains the following data.

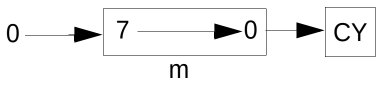
7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0

Upon the execution of an SRA *(IX*+`3h`*)* instruction, memory location `1003h` and the Carry flag now contain:

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	0	0

SRL m

Operation



Op Code

SRL

Operand

m

The operand *m* is any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*, as defined for the analogous RLC instructions. In the assembled object code, the possible op code/operand combinations are specified as follows:

SRL r*	1	1	0	0	1	0	1	1	CB
	0	0	1	1	1	◄—	r*	—►	
SRL (HL)	1	1	0	0	1	0	1	1	CB
	0	0	1	1	1	1	1	0	3E
SRL (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	◄—				d			—►	
	0	0	1	1	1	1	1	0	3E
SRL (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	◄—				d			—►	
	0	0	1	1	1	1	1	0	3E

r identifies registers B, C, D, E, H, L, or A.

Description

The contents of operand *m* are shifted right 1 bit position. The contents of bit 0 are copied to the Carry flag, and bit 7 is reset. Bit 0 is the least-significant bit.

Instruction	M Cycles	T States	4 MHz E.T.
SRL r	2	8 (4, 4)	2.00
SRL (HL)	4	15 (4, 4, 4, 3)	3.75
SRL (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
SRL (lY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

S is reset.

Z is set if result is 0; otherwise, it is reset.

H is reset.

P/V is set if parity is even; otherwise, it is reset.

N is reset.

C is data from bit 0 of source register.

Example

Register B contains the following data.

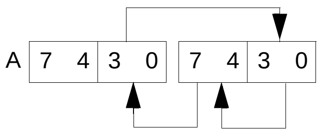
7	6	5	4	3	2	1	0
1	0	0	0	1	1	1	1

Upon the execution of an SRL *B* instruction, Register B and the Carry flag now contain:

7	6	5	4	3	2	1	0	C
0	1	0	0	0	1	1	1	1

RLD

Operation



Op Code

RLD

Operands

1	1	1	0	1	1	0	1	ED
0	1	1	0	1	1	1	1	6F

Description

The contents of the low-order four bits (bits 3, 2, 1, and 0) of the memory location (HL) are copied to the high-order four bits (7, 6, 5, and 4) of that same memory location; the previous contents of those high-order four bits are copied to the low-order four bits of the Accumulator (Register A); and the previous contents of the low-order four bits of the Accumulator are copied to the low-order four bits of memory location (HL). The contents of the high-order bits of the Accumulator are unaffected.

Note: (HL) refers to the memory location specified by the contents of the HL register pair.

M Cycles	T States	4 MHz E.T.
5	18 (4, 4, 3, 4, 3)	4.50

Condition Bits Affected

S is set if the Accumulator is negative after an operation; otherwise, it is reset.

Z is set if the Accumulator is 0 after an operation; otherwise, it is reset.

H is reset.

P/V is set if the parity of the Accumulator is even after an operation; otherwise, it is reset.

N is reset.

C is not affected.

Example

The HL register pair contains `5000h` and the Accumulator and memory location `5000h` contain the following data.

7	6	5	4	3	2	1	0	
0	1	1	1	1	0	1	0	Accumulator

7	6	5	4	3	2	1	0	
0	0	1	1	0	0	0	1	(5000h)

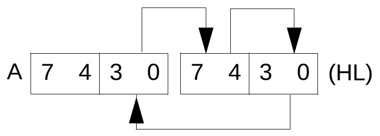
Upon the execution of an RLD instruction, the Accumulator and memory location `5000h` now contain:

7	6	5	4	3	2	1	0	
0	1	1	1	0	0	1	1	Accumulator

7	6	5	4	3	2	1	0	
0	0	0	1	1	0	1	0	(5000h)

RRD

Operation



Op Code

RRD

Operands

1	1	1	0	1	1	0	1	ED
0	1	1	0	0	1	1	1	67

Description

The contents of the low-order four bits (bits 3, 2, 1, and 0) of memory location (HL) are copied to the low-order four bits of the Accumulator (Register A). The previous contents of the low-order four bits of the Accumulator are copied to the high-order four bits (7, 6, 5, and 4) of location (HL); and the previous contents of the high-order four bits of (HL) are copied to the low-order four bits of (HL). The contents of the high-order bits of the Accumulator are unaffected.

Note: (HL) refers to the memory location specified by the contents of the HL register pair.

M Cycles	T States	4 MHz E.T.
5	18 (4, 4, 3, 4, 3)	4.50

Condition Bits Affected

S is set if the Accumulator is negative after an operation; otherwise, it is reset.

Z is set if the Accumulator is 0 after an operation; otherwise, it is reset.

H is reset.

P/V is set if the parity of the Accumulator is even after an operation; otherwise, it is reset.

N is reset.

C is not affected.

Example

The HL register pair contains `5000h` and the Accumulator and memory location `5000h` contain the following data.

7	6	5	4	3	2	1	0	
1	0	0	0	0	1	0	0	Accumulator

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	(5000h)

Upon the execution of an RRD instruction, the Accumulator and memory location `5000h` now contain:

7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	Accumulator

7	6	5	4	3	2	1	0	
0	1	0	0	0	0	1	0	(5000h)

Bit Set, Reset, and Test Group

The following bit set, reset, and test group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

BIT b, r

Operation

$Z \leftarrow \overline{rb}$

Op Code

BIT

Operands

b, *r*

1	1	0	0	1	0	1	1	CB
0	1	◄	b	►	◄	r	►	

Description

This instruction tests bit *b* in register *r* and sets the Z flag accordingly. In the assembled object code, operands *b* and *r* are specified as follows:

Bit Tested	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	4.50

Condition Bits Affected

S is unknown.

Z is set if specified bit is 0; otherwise, it is reset.

H is set.

P/V is unknown.

N is reset.

C is not affected.

Example

If bit 2 in Register B contains 0, then upon the execution of a BIT *2*, *B* instruction, the Z flag in the F Register contains 1, and bit 2 in Register B remains at 0. Bit 0 in Register B is the least-significant bit.

BIT b, (HL)

Operation

Z ← (HL)b

Op Code

BIT

Operands

b, (HL)

1	1	0	0	1	0	1	1	CB
0	1		b		1	1	0	

Description

This instruction tests bit b in the memory location specified by the contents of the HL register pair and sets the Z flag accordingly. In the assembled object code, operand *b* is specified as follows:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
1	111

M Cycles	T States	4 MHz E.T.
3	12 (4, 4, 4) 4	3.00

Condition Bits Affected

S is unknown.

Z is set if specified bit is 0; otherwise, it is reset.

H is set.

P/V is unknown.

H is reset.

C is not affected.

Example

If the HL register pair contains `4444h`, and bit 4 in the memory location `444h` contains 1, then upon the execution of a BIT *4, (HL)* instruction, the Z flag in the F Register contains 0, and bit 4 in memory location `4444h` remains at 1. Bit 0 in memory location `4444h` is the least-significant bit.

BIT b, (IX+d)

Operation

Z ← ($\overline{\text{IX+d}}$)b

Op Code

BIT

Operands

b, (IX+d)

1	1	0	1	1	1	0	1	DD
1	1	0	0	1	0	1	1	CB
←			d				→	
0	1	←	b	→	1	1	0	

Description

This instruction tests bit b in the memory location specified by the contents of register pair IX combined with the two's complement displacement d and sets the Z flag accordingly. In the assembled object code, operand *b* is specified as follows:

Bit Tested	**b**
0	`000`
1	`001`
2	`010`
3	`011`
4	`100`
5	`101`
6	`110`
7	`111`

M Cycles	**T States**	**4 MHz E.T.**
5	20 (4, 4, 3, 5, 4)	5.00

Condition Bits Affected

S is unknown.

Z is set if specified bit is 0; otherwise, it is reset.

H is set.

P/V is unknown.

N is reset.

C is not affected.

Example

If Index Register IX contains `2000h` and bit 6 in memory location `2004h` contains 1, then upon the execution of a BIT *6, (IX+*`4h`*)* instruction, the Z flag in the F Register contains a 0 and bit 6 in memory location `2004h` still contains a 1. Bit 0 in memory location `2004h` is the least-significant bit.

BIT b, (IY+d)

Operation

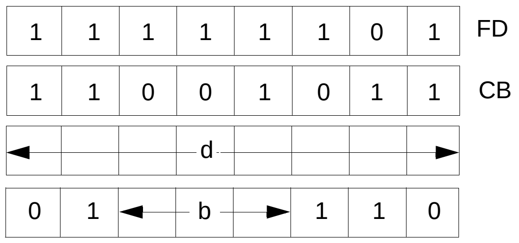
Z ← $\overline{\text{(IY+d)}}$b

Op Code

BIT

Operands

b, (lY+d)



Description

This instruction tests bit b in the memory location specified by the contents of register pair IY combined with the two's complement displacement *d* and sets the Z flag accordingly. In the assembled object code, operand *b* is specified as follows.

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
5	20 (4, 4, 3, 5, 4)	5.00

Condition Bits Affected

S is unknown.

Z is set if specified bit is 0; otherwise, it is reset.

H is set.

P/V is unknown.

H is reset.

C is not affected.

Example

If Index Register contains `2000h` and bit 6 in memory location `2004h` contains a 1, then upon the execution of a BIT *6, (IY+*`4h`*)* instruction, the Z flag and the F Register still contains a 0, and bit 6 in memory location `2004h` still contains a 1. Bit 0 in memory location `2004h` is the least-significant bit.

SET b, r

Operation

rb ← 1

Op Code

SET

Operands

b, r

1	1	0	0	1	0	1	1	CB
1	1	◄—	b	—►	◄—	r	—►	

Description

Bit b in register *r* (any of registers B, C, D, E, H, L, or A) is set. In the assembled object code, operands *b* and *r* are specified as follows:

Bit	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected

None.

Example

Upon the execution of a SET *4, A* instruction, bit 4 in Register A is set. Bit 0 is the least-significant bit.

SET b, (HL)

Operation

(HL)b ← 1

Op Code

SET

Operands

b, (HL)

1	1	0	0	1	0	1	1	CB

1	1	◄—	b	—►	◄—	r	—►

Description

Bit b in the memory location addressed by the contents of register pair HL is set. In the assembled object code, operand *b* is specified as follows:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M Cycles	T States	4 MHz E.T.
4	15 (4, 4, 4, 3)	3.75

Condition Bits Affected

None.

Example

If the HL register pair contains `3000h`, then upon the execution of a SET *4*, *(HL)* instruction, bit 4 in memory location `3000h` is 1. Bit 0 in memory location `3000h` is the least-significant bit.

SET b, (IX+d)

Operation

(IX+d)b ← 1

Op Code

SET

Operands

b, (IX+d)

Description

Bit b in the memory location addressed by the sum of the contents of the IX register pair and the two's complement integer *d* is set. In the assembled object code, operand *b* is specified as follows:

Bit Tested	b
0	`000`
1	`001`
2	`010`
3	`011`
4	`100`
5	`101`
6	`110`
7	`111`

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

None.

Example

If the index register contains `2000h`, then upon the execution of a SET *0, (IX +* `3h`*)* instruction, bit 0 in memory location `2003h` is 1. Bit 0 in memory location `2003h` is the least-significant bit.

SET b, (IY+d)

Operation

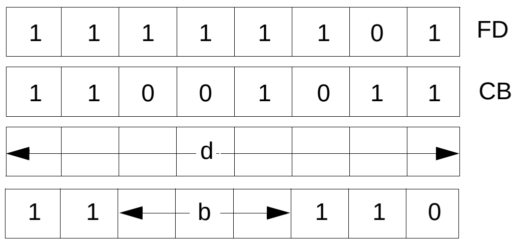
(IY + d) b ← 1

Op Code

SET

Operands

b, (IY + d)



Description

Bit b in the memory location addressed by the sum of the contents of the IY register pair and the two’s complement displacement d is set. In the assembled object code, operand *b* is specified as follows:

Bit Tested	b
0	`000`
1	`001`
2	`010`
3	`011`
4	`100`
5	`101`
6	`110`
7	`111`

M Cycles	T States	4 MHz E.T.
6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

None.

Example

If Index Register IY contains `2000h`, then upon the execution of a Set *0, (IY+*`3h`*)* instruction, bit 0 in memory location `2003h` is 1. Bit 0 in memory location `2003h` is the least-significant bit.

RES b, m

Operation

sb ← 0

Op Code

RES

Operands

b, m

The *b* operand represents any bit (7 through 0) of the contents of the *m* operand, (any of *r*, *(HL)*, *(IX+d)*, or *(lY+d)*) as defined for the analogous SET instructions. These possible op code/operand combinations are assembled as follows in the object code:

RES b, rn	1	1	0	0	1	0	1	1	CB
	1	0	←	b	→	←	r	→	
RES b, (HL)	1	1	0	0	1	0	1	1	CB
	1	0	←	b	→	1	1	0	
RES b, (IX+d)	1	1	0	1	1	1	0	1	DD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	1	0	←	b	→	1	1	0	
RES b, (IY+d)	1	1	1	1	1	1	0	1	FD
	1	1	0	0	1	0	1	1	CB
	←			d				→	
	1	0	←	b	→	1	1	0	

Bit	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

Description

Bit b in operand *m* is reset.

Instruction	M Cycles	T States	4 MHz E.T.
RES r	4	8 (4, 4)	2.00
RES (HL)	4	15 (4, 4, 4, 3)	3.75
RES (IX+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75
RES (lY+d)	6	23 (4, 4, 3, 5, 4, 3)	5.75

Condition Bits Affected

None.

Example

Upon the execution of a RES *6, D* instruction, bit 6 in register 0 is reset. Bit 0 in the D Register is the least-significant bit.

Jump Group

The following jump group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

JP nn

Operation

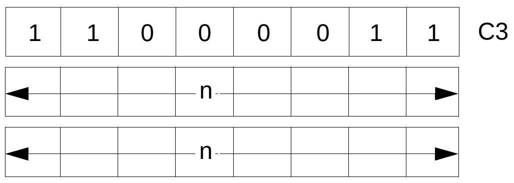
PC ← nn

Op Code

JP

Operand

nn



Note: The first operand in this assembled object code is the low-order byte of a two-byte address.

Description

Operand *nn* is loaded to register pair Program Counter (PC). The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

JP cc, nn

Operation

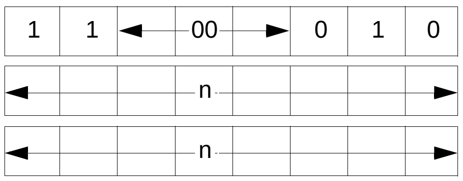
IF cc true, PC ← nn

Op Code

JP

Operands

cc, *nn*



The first *n* operand in this assembled object code is the low-order byte of a 2-byte memory address.

Description

If condition *cc* is true, the instruction loads operand *nn* to register pair Program Counter (PC), and the program continues with the instruction beginning at address *nn*. If condition *cc* is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. Condition *cc* is programmed as one of eight statuses that correspond to condition bits in the Flag Register (Register F). These eight statuses are defined in the following table, which specifies the corresponding *cc* bit fields in the assembled object code.

cc	Condition	Relevant Flag
000	Non-Zero (NZ)	Z
001	Zero (Z)	Z
010	No Carry (NC)	C
011	Carry (C)	C
100	Parity Odd (PO)	P/V
101	Parity Even (PE)	P/V
110	Sign Positive (P)	S
111	Sign Negative (M)	S

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

Example

If the Carry flag (i.e., the C flag in F Register) is set and address `1520h` contains `03h`, then upon the execution of a JP *C*, `1520h` instruction, the Program Counter contains `1520h` and, on the next machine cycle, the CPD fetches byte `03h` from address `1520h`.

JR e

Operation

PC ← PC + e

Op Code

JR

Operand

e

0	0	0	1	1	0	0	0	18
◄			e–2				►	

Description

This instruction provides for unconditional branching to other segments of a program. The value of displacement *e* is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction op code and contains a range of –126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

Condition Bits Affected

None.

Example

To jump forward five locations from address 480, the following assembly language statement is used:

```
JR $+5
```

The resulting object code and final Program Counter value is shown in the following table:

Location	Instruction
480	18
481	03
482	–
483	–
484	–
485	← PC after jump

JR C, e

Operation

If C = 0, continue
If C = 1, PC ← PC+ e

Op Code

JR

Operands

C, *e*

0	0	1	1	1	0	0	0	38
◄				e–2			►	

Description

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag = 1, the value of displacement *e* is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction op code and contains a range of –126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag = 0, the next instruction executed is taken from the location following this instruction. If condition is met

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If condition is not met:

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

The Carry flag is set and it is required to jump back four locations from 480. The assembly language statement is `JR C, $–4`

The resulting object code and final Program Counter value is shown in the following table:

Location	Instruction
47C	← PC after jump
47D	–
47E	–
47F	–
480	38
481	FA (two's complement – 6)

JR NC, e

Operation

If C = 1, continue
If C = 0, PC ← PC + e

Op Code

JR

Operands

NC, e

0	0	1	1	0	0	0	0	30
◄			e–2				►	

Description

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to 0, the value of displacement *e* is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction op code and contains a range of –126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag = 1, the next instruction executed is taken from the location following this instruction.

If the condition is met:

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If the condition is not met:

M Cycles	T States	4 MHz E.T.
7	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

The Carry Flag is reset and it is required to repeat the jump instruction. The assembly language statement is `JR NC, $`

The resulting object code and Program Counter after the jump are:

Location	Instruction
480	30 ← PC after jump
481	00

JR Z, e

Operation

If Z = 0, continue
If Z = 1, PC ← PC + e

Op Code

JR

Operands

Z, e

0	0	1	0	1	0	0	0	28
◄			e–2				►	

Description

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag = 1, the value of displacement *e* is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction op code and contains a range of –126 to +129 bytes. The assembler automatically adjusts for the twice-incremented PC.

If the Zero Flag = 0, the next instruction executed is taken from the location following this instruction.

If this condition is met, the following data results:

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If this condition is not met, the following data results:

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

The Zero Flag is set and it is required to jump forward five locations from address 300. The following assembly language statement is used:

```
JR Z ,$ + 5
```

The resulting object code and final Program Counter value are:

Location	Instruction
300	28
301	03
302	–
303	–
304	–
305	← PC after jump

JR NZ, e

Operation

If Z = 1, continue
If Z = 0, PC ← pc + e

Op Code

JR

Operands

NZ, e

0	0	1	0	0	0	0	0	20
◄			e–2				►	

Description

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag = 0, the value of displacement *e* is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction op code and contains a range of –126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag = 1, the next instruction executed is taken from the location following this instruction.

If the condition is met:

M Cycles	T States	4 MHz E.T.
3	12 (4, 3, 5)	3.00

If the condition is not met:

M Cycles	T States	4 MHz E.T.
2	7 (4, 3)	1.75

Condition Bits Affected

None.

Example

The Zero Flag is reset and it is required to jump back four locations from 480. The assembly language statement is `JR NZ`, $–4

The resulting object code and final Program Counter value is:

Location	Instruction
47C	← PC after jump
47D	–
47E	–
47F	–
480	20
481	FA (two's complement – 6)

JP (HL)

Operation

PC ← HL

Op Code

JP

Operand

(HL)

1	1	1	0	1	0	0	1	E9

Description

The Program Counter (PC) is loaded with the contents of the HL register pair. The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
1	4	1.00

Condition Bits Affected

None.

Example

If the Program Counter contains `1000h` and the HL register pair contains `4800h`, then upon the execution of a JP *(HL)* instruction, the Program Counter contains `4800h`.

JP (IX)

Operation

pc ← IX

Op Code

JP

Operand

(IX)

1	1	0	1	1	1	0	1	DD
1	1	1	0	1	0	0	1	E9

Description

The Program Counter (PC) is loaded with the contents of the IX register pair. The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected

None.

Example

If the Program Counter contains `1000h` and the IX register pair contains `4800h`, then upon the execution of a JP *(IX)* instruction, the Program Counter contains `4800h`.

JP (IY)

Operation

PC ← IY

Op Code

JP

Operand

(IY)

1	1	1	1	1	1	0	1	FD
1	1	1	0	1	0	0	1	E9

Description

The Program Counter (PC) is loaded with the contents of the IY register pair. The next instruction is fetched from the location designated by the new contents of the PC.

M Cycles	T States	4 MHz E.T.
2	8 (4, 4)	2.00

Condition Bits Affected

None.

Example

If the Program Counter contains `1000h` and the IY register pair contains `4800h`, then upon the execution of a JP *(IY)* instruction, the Program Counter contains `4800h`.

DJNZ, e

Operation

B ← B – 1

If B = 0, continue

If B ≠ 0, PC ← PC + e

Op Code

DJNZ

Operand

e

0	0	0	1	0	0	0	0	10
◄			e–2				►	

Description

This instruction is similar to the conditional jump instructions except that a register value is used to determine branching. Register B is decremented, and if a nonzero value remains, the value of displacement *e* is added to the Program Counter (PC). The next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction op code and contains a range of –126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the result of decrementing leaves B with a zero value, the next instruction executed is taken from the location following this instruction.

if B ≠ 0:

M Cycles	T States	4 MHz E.T.
3	13 (5,3, 5)	3.25

If B = 0:

M Cycles	T States	4 MHz E.T.
2	8 (5, 3)	2.00

Condition Bits Affected

None.

Example

A typical software routine is used to demonstrate the use of the DJNZ instruction. This routine moves a line from an input buffer (INBUF) to an output buffer (OUTBUF). It moves the bytes until it finds a CR, or until it has moved 80 bytes, whichever occurs first.

```
      LD          8, 80       ;Set up counter
      LD          HL, Inbuf   ;Set up pointers
      LD          DE, Outbuf

LOOP: LID         A, (HL)     ;Get next byte from
                              ;input buffer
      LD          (DE), A     ;Store in output buffer
      CP          ODH         ;Is it a CR?
      JR          Z, DONE     ;Yes finished
      INC         HL          ;Increment pointers
      INC         DE
      DJNZ LOOP               ;Loop back if 80
                              ;bytes have not
                              ;been moved
DONE:
```

Call and Return Group

The following call and return group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

CALL nn

Operation

(SP – 1) ← PCH, (SP – 2) ← PCL, PC ← nn

Op Code

CALL

Operand

nn

1	1	0	0	1	1	0	1	CD
◄			n				►	
◄			n				►	

The first of the two *n* operands in the assembled object code above is the least-significant byte of a 2-byte memory address.

Description

The current contents of the Program Counter (PC) are pushed onto the top of the external memory stack. The operands *nn* are then loaded to the PC to point to the address in memory at which the first op code of a subroutine is to be fetched. At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back to the PC. The push is accomplished by first decrementing the current contents of the Stack Pointer (register pair SP), loading the high-order byte of the PC contents to the memory address now pointed to by the SP; then decrementing SP again, and loading the low-order byte of the PC contents to the top of stack.

Because this process is a 3-byte instruction, the Program Counter was incremented by three before the push is executed.

M Cycles	T States	4 MHz E.T.
5	17 (4, 3, 4, 3, 3)	4.25

Condition Bits Affected

None.

Example

The Program Counter contains `1A47h`, the Stack Pointer contains `3002h`, and memory locations contain the following data.

Location	Contents
1A47h	CDh
IA48h	35h
1A49h	21h

If an instruction fetch sequence begins, the 3-byte instruction CD `3521h` is fetched to the CPU for execution. The mnemonic equivalent of this instruction is CALL `2135h`. Upon the execution of this instruction, memory address `3001h` contains `1Ah`, address `3000h` contains `4Ah`, the Stack Pointer contains `3000h`, and the Program Counter contains `2135h`, thereby pointing to the address of the first op code of the next subroutine to be executed.

CALL cc, nn

Operation

IF cc true: (sp – 1) ← PCH

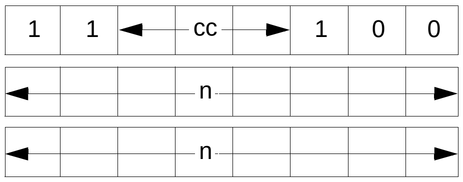
(sp – 2) ← PCL, pc ← nn

Op Code

CALL

Operands

cc, nn



The first of the two *n* operands in the assembled object code above is the least-significant byte of the 2-byte memory address.

Description

If condition *cc* is true, this instruction pushes the current contents of the Program Counter (PC) onto the top of the external memory stack, then loads the operands *nn* to PC to point to the address in memory at which the first op code of a subroutine is to be fetched. At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back to PC. If condition *cc* is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. The stack push is accomplished by first decrementing the current contents of the Stack Pointer (SP), loading the high-order byte of the PC contents to the memory address now pointed to by SP; then decrementing SP again, and loading the low-order byte of the PC contents to the top of the stack.

Because this process is a 3-byte instruction, the Program Counter was incremented by three before the push is executed.

Condition *cc* is programmed as one of eight statuses that corresponds to condition bits in the Flag Register (Register F). These eight statuses are defined in the following table, which also specifies the corresponding *cc* bit fields in the assembled object code.

cc	Condition	Relevant Flag
`000`	Non-Zero (NZ)	Z
`001`	Zero (Z)	Z
`010`	Non Carry (NC)	C
`011`	Carry (C)	Z
`100`	Parity Odd (PO)	P/V
`101`	Parity Even (PE)	P/V
`110`	Sign Positive (P)	S
`111`	Sign Negative (M)	S

If *cc* is true:

M Cycles	T States	4 MHz E.T.
5	17 (4, 3, 4, 3, 3)	4.25

If *cc* is false:

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

Example

The C Flag in the F Register is reset, the Program Counter contains `1A47h`, the Stack Pointer contains `3002h`, and memory locations contain the following data.

Location	Contents
1A47h	D4h
1448h	35h
1A49h	21h

If an instruction fetch sequence begins, the 3-byte instruction `D43521h` is fetched to the CPU for execution. The mnemonic equivalent of this instruction is CALL *NC*, `2135h`. Upon the execution of this instruction, memory address `3001h` contains `1Ah`, address `3000h` contains `4Ah`, the Stack Pointer contains `3000h`, and the Program Counter contains `2135h`, thereby pointing to the address of the first op code of the next subroutine to be executed.

RET

Operation

pCL ← (sp), pCH ← (sp+1)

Op Code

RET

1	1	0	0	1	0	0	1	C9

Operands

None.

Description

The byte at the memory location specified by the contents of the Stack Pointer (SP) Register pair is moved to the low-order eight bits of the Program Counter (PC). The SP is now incremented and the byte at the memory location specified by the new contents of this instruction is fetched from the memory location specified by the PC. This instruction is normally used to return to the main line program at the completion of a routine entered by a CALL instruction.

M Cycles	T States	4 MHz E.T.
3	10 (4, 3, 3)	2.50

Condition Bits Affected

None.

Example

The Program Counter contains `3535h`, the Stack Pointer contains `2000h`, memory location `2000h` contains `B5h`, and memory location `2001h` contains `18h`. Upon the execution of a RET instruction, the Stack Pointer contains `2002h` and the Program Counter contains `18B5h`, thereby pointing to the address of the next program op code to be fetched.

RET cc

Operation

If cc true: PCL ← (sp), pCH ← (sp+1)

Op Code

RET

Operand

cc

1	1	◄	cc	►	0	0	0

Description

If condition *cc* is true, the byte at the memory location specified by the contents of the Stack Pointer (SP) Register pair is moved to the low-order eight bits of the Program Counter (PC). The SP is incremented and the byte at the memory location specified by the new contents of the SP are moved to the high-order eight bits of the PC. The SP is incremented again. The next op code following this instruction is fetched from the memory location specified by the PC. This instruction is normally used to return to the main line program at the completion of a routine entered by a CALL instruction. If condition *cc* is false, the PC is simply incremented as usual, and the program continues with the next sequential instruction. Condition *cc* is programmed as one of eight status that correspond to condition bits in the Flag Register (Register F). These eight status are defined in the following table, which also specifies the corresponding *cc* bit fields in the assembled object code.

cc	Condition	Relevant Flag
`000`	Non-Zero (NZ)	Z
`001`	Zero (Z)	Z
`010`	Non Carry (NC)	C
`011`	Carry (C)	C
`100`	Parity Odd (PO)	P/V
`101`	Parity Even (PE)	P/V
`110`	Sign Positive (P)	S
`111`	Sign Negative (M)	S

If *cc* is true, then the following data is returned:

M Cycles	T States	4 MHz E.T.
3	11 (5, 3, 3)	2.75

If *cc* is false, then the following data is returned:

M Cycles	T States	4 MHz E.T.
1	5	1.25

Condition Bits Affected

None.

Example

The S flag in the F Register is set, the Program Counter contains `3535h`, the Stack Pointer contains `2000h`, memory location `2000h` contains `B5h`, and memory location `2001h` contains `18h`. Upon the execution of a RET *M* instruction, the Stack Pointer contains `2002h` and the Program Counter contains `18B5h`, thereby pointing to the address of the next program op code to be fetched.

RETI

Operation

Return from Interrupt

Op Code

RETI

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	0	1	4D

Operands

None.

Description

This instruction is used at the end of a maskable interrupt service routine to:

- Restore the contents of the Program Counter (analogous to the RET instruction)
- Signal an I/O device that the interrupt routine is completed. The RETI instruction also facilitates the nesting of interrupts, allowing higher priority devices to temporarily suspend service of lower priority service routines. However, this instruction does not enable interrupts that were disabled when the interrupt routine was entered. Before doing the RETI instruction, the enable interrupt instruction (EI) should be executed to allow recognition of interrupts after completion of the current service routine.

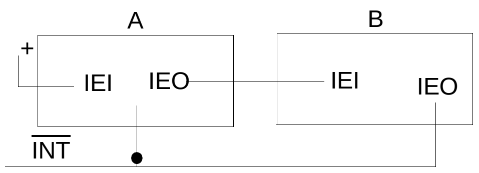
M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected

None.

Example

Assume that there are two interrupting devices, A and B, connected in a daisy-chain configuration, with A having a higher priority than B.



B generates an interrupt and is acknowledged. The interrupt enable out, IEO, of B goes Low, blocking any lower priority devices from interrupting while B is being serviced. Then A generates an interrupt, suspending service of B. The IEO of A goes Low, indicating that a higher priority device is being serviced. The A routine is completed and a RETI is issued resetting the IEO of A, allowing the B routine to continue. A second RETI is issued on completion of the B routine and the IE0 of B is reset (High), allowing lower-priority devices interrupt access.

RETN

Operation

Return from nonmaskable interrupt

Op Code

RETN

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	1	45

Operands

None.

Description

This instruction is used at the end of a nonmaskable interrupts service routine to restore the contents of the Program Counter (analogous to the RET instruction). The state of IFF2 is copied back to IFF1 so that maskable interrupts are enabled immediately following the RETN if they were enabled before the nonmaskable interrupt.

M Cycles	T States	4 MHz E.T.
4	14 (4, 4, 3, 3)	3.50

Condition Bits Affected

None.

Example

If the Stack Pointer contains `1000h` and the Program Counter contains `1A45h` when a Nonmaskable Interrupt (NMI) signal is received, the CPU ignores the next instruction and instead restarts, returning to memory address `0066h`. The current Program Counter contains `1A45h`, which is pushed onto the external stack address of `0FFFh` and `0FFEh`, high-order byte first, and `0066h` is loaded onto the Program Counter. That address begins an interrupt service routine that ends with a RETN instruction.

Upon the execution of a RETN instruction, the contents of the former Program Counter are popped off the external memory stack, low-order first, resulting in the Stack Pointer again containing `1000h`. The program flow continues where it left off with an op code fetch to address `1A45h`, order-byte first, and `0066h` is loaded onto the Program Counter.

That address begins an interrupt service routine that ends with a RETN instruction. Upon the execution of a RETN instruction, the contents of the former Program Counter are popped off the external memory stack, low-order first, resulting in stack pointer contents of `1000h`. The program flow continues where it left off with an op code fetch to address `1A45h`.

RST p

Operation

(SP – 1) ← PCH, (SP – 2) ← PCL, PCH ← 0, PCL ← P

Op Code

RST

Operand

p

1	1	◄—	t	—►	1	1	1

Description

The current Program Counter (PC) contents are pushed onto the external memory stack, and the Page 0 memory location assigned by operand *p* is loaded to the PC. Program execution then begins with the op code in the address now pointed to by PC. The push is performed by first decrementing the contents of the Stack Pointer (SP), loading the high-order byte of PC to the memory address now pointed to by SP, decrementing SP again, and loading the low-order byte of PC to the address now pointed to by SP. The Restart instruction allows for a jump to one of eight addresses indicated in the following table. The operand *p* is assembled to the object code using the corresponding T state.

Because all addresses are stored in Page 0 of memory, the high-order byte of PC is loaded with `00h`. The number selected from the *p* column of the table is loaded to the low-order byte of PC.

p	t
00h	000
08h	001
10h	010
18h	011
20h	100
28h	101
30h	110
38h	111

M Cycles	T States	4 MHz E.T.
3	11 (5, 3, 3)	2.75

Example

If the Program Counter contains `15B3h`, then upon the execution of an RST `18h` (object code `1101111`) instruction, the PC contains `0018h` as the address of the next fetched op code.

Input and Output Group

The following input and output group instructions are each described in this section. Simply click to jump to an instruction's description to learn more.

IN A, (n)

Operation

A ← (n)

Op Code

IN

Operands

A, (n)

1	1	0	1	1	0	1	1	DB
◄			n				►	

Description

The operand *n* is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator also appear on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the Accumulator (Register A) in the CPU.

M Cycles	T States	4 MHz LT.
3	11 (4, 3, 4)	2.75

Condition Bits Affected

None.

Example

The Accumulator contains `23h`, and byte `7Bh` is available at the peripheral device mapped to I/O port address `01h`. Upon the execution of an IN *A*, (`01h`) instruction, the Accumulator contains `7Bh`.

IN r (C)

Operation

r ← (C)

Op Code

IN

Operands

r, (C)

1	1	1	0	1	1	0	1	EB
0	1	◄	r	►	0	0	0	

Description

The contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to register *r* in the CPU. Register *r* identifies any of the CPU registers shown in the following table, which also indicates the corresponding 3-bit *r* field for each. The flags are affected, checking the input data.

Register	r	
Flag	`110`	Undefined op code; set the flag
B	`000`	
C	`001`	
D	`010`	
E	`011`	
H	`100`	
L	`101`	
A	`111`	

M Cycles	T States	4 MHz E.T.
3	12 (4, 4, 4)	3.00

Condition Bits Affected

S is set if input data is negative; otherwise, it is reset.

Z is set if input data is 0; otherwise, it is reset.

H is reset.

P/V is set if parity is even; otherwise, it is reset.

N is reset.

C is not affected.

Example

Register C contains `07h`, Register B contains `10h`, and byte `7Bh` is available at the peripheral device mapped to I/O port address `07h`. Upon the execution of an IN *D*, *(C)* command, the D Register contains `7Bh`.

INI

Operation

(HL) ← (C), B ← B – 1, HL ← HL + 1

Op Code

INI

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	0	A2

Operands

None.

Description

The contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B can be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are then placed on the address bus and the input byte is written to the corresponding location of memory. Finally, the byte counter is decremented and register pair HL is incremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected

S is unknown.

Z is set if B – 1 = 0; otherwise it is reset.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

Register C contains `07h`, Register B contains `10h`, the HL register pair contains `1000h`, and byte `7Bh` is available at the peripheral device mapped to I/O port address `07h`. Upon the execution of an INI instruction, memory location `1000h` contains `7Bh`, the HL register pair contains `1001h`, and Register B contains `0Fh`.

INIR

Operation

(HL) ← (C), B ← B – 1, HL ← HL +1

Op Code

INIR

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	1	0	B2

Operands

None.

Description

The contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written to the corresponding location of memory. Then register pair HL is incremented, the byte counter is decremented. If decrementing causes B to go to 0, the instruction is terminated. If B is not 0, the Program Counter is decremented by two and the instruction repeated. Interrupts are recognized and two refresh cycles execute after each data transfer.

Note: If B is set to 0 prior to instruction execution, 256 bytes of data are input.

If B ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25

If B = 0:

M Cycles	T States	4	MHz E.T.
4	16 (4, 5, 3, 4)	4.00	

Condition Bits Affected

S is unknown.

Z is set.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

Register C contains `07h`, Register B contains `03h`, the HL register pair contains `1000h`, and the following sequence of bytes is available at the peripheral device mapped to I/O port of address `07h`.

```
51h
A9h
03h
```

Upon the execution of an INIR instruction, the HL register pair contains `1003h`, Register B contains a 0, and the memory locations contain the following data:

1000h	51h
1001h	A9h
1002h	03h

IND

Operation

(HL) ← (C), B ← B – 1, HL ← HL – 1

Op Code

IND

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	1	0	AA

Operands

None.

Description

The contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B can be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written to the corresponding location of memory. Finally, the byte counter and register pair HL are decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected

S is unknown.

Z is set if B – 1 = 0; otherwise, it is reset.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

Register C contains `07h`, Register B contains `10h`, the HL register pair contains `1000h`, and byte `7Bh` is available at the peripheral device mapped to I/O port address `07h`. Upon the execution of an IND instruction, memory location `1000h` contains `7Bh`, the HL register pair contains `0FFFh`, and Register B contains `0Fh`.

INDR

Operation

(HL) ← (C), B ← 131, HL ← HL1

Op Code

INDR

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	1	0	BA

Operands

None.

Description

The contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written to the corresponding location of memory. Then HL and the byte counter are decremented. If decrementing causes B to go to 0, the instruction is terminated. If B is not 0, the Program Counter is decremented by two and the instruction repeated. Interrupts are recognized and two refresh cycles are executed after each data transfer.

When B is set to 0 prior to instruction execution, 256 bytes of data are input.

If B ≠ 0:

M Cycles	T States	4	MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25	

If B = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected

S is unknown.

Z is set.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

Register C contains `07h`, Register B contains `03h`, the HL register pair contains `1000h` and the following sequence of bytes is available at the peripheral device mapped to I/O port address `07h`:

```
51h
A9h
03h
```

Upon the execution of an INDR instruction, the HL register pair contains `0FFDh`, Register B contains a 0, and the memory locations contain the following data:

0FFEh	03h
0FFFh	A9h
1000h	51h

OUT (n), A

Operation

(n) ← A

Op Code

OUT

Operands

(n), A

1	1	0	1	0	0	1	1	D3
←			n				→	

Description

The operand *n* is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator (Register A) also appear on the top half (A8 through A15) of the address bus at this time. Then the byte contained in the Accumulator is placed on the data bus and written to the selected peripheral device.

M Cycles	T States	4 MHz E.T.
3	11 (4, 3, 4)	2.75

Condition Bits Affected

None.

Example

If the Accumulator contains `23h`, then upon the execution of an OUT (`01h`) instruction, byte `23h` is written to the peripheral device mapped to I/O port address `01h`.

OUT (C), r

Operation

(C) ← r

Op Code

OUT

Operands

(C), r

1	1	1	0	1	1	0	1	ED
0	1	◄—	r	—►	0	0	1	

Description

The contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A8 through A15) of the address bus at this time. Then the byte contained in register *r* is placed on the data bus and written to the selected peripheral device. Register *r* identifies any of the CPU registers shown in the following table, which also shows the corresponding three-bit *r* field for each that appears in the assembled object code.

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M Cycles	T States	4 MHz E.T.
3	12 (4, 4, 4)	3.00

Condition Bits Affected

None.

Example

If Register C contains `01h` and the D Register contains `5Ah`, then upon the execution of an OUT *(C), D* instruction, byte `5Ah` is written to the peripheral device mapped to I/O port address `01h`.

OUTI

Operation

(C) ← (HL), B ← B – 1, HL ← HL + 1

Op Code

OUTI

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	1	A3

Operands

None.

Description

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B can be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus. The byte to be output is placed on the data bus and written to a selected peripheral device. Finally, the register pair HL is incremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected

S is unknown.

Z is set if B – 1 = 0; otherwise, it is reset.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

If Register C contains `07h`, Register B contains `10h`, the HL register pair contains `100014` and memory address `1000h` contains `5914`, then upon the execution of an OUTI instruction, Register B contains `0Fh`, the HL register pair contains `1001h`, and byte `59h` is written to the peripheral device mapped to I/O port address `07h`.

OTIR

Operation

(C) ← (HL), B ← B – 1, HL ← HL + 1

Op Code

OTIR

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	1	1	B3

Operands

None.

Description

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B can be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next, the byte to be output is placed on the data bus and written to the selected peripheral device. Then register pair HL is incremented. If the decremented B Register is not 0, the Program Counter (PC) is decremented by two and the instruction is repeated. If B has gone to 0, the instruction is terminated. Interrupts are recognized and two refresh cycles are executed after each data transfer.

Note: When B is set to 0 prior to instruction execution, the instruction outputs 256 bytes of data.

If B ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25

If B = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected

S is unknown.

Z is set.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

Register C contains `07h`, Register B contains `03h`, the HL register pair contains `1000h`, and memory locations contain the following data.

1000h	contains 51h
1001h	contains A9h
1002h	contains 03h

Upon the execution of an OTIR instruction, the HL register pair contains `1003h`, Register B contains a 0, and a group of bytes is written to the peripheral device mapped to I/O port address `07h` in the following sequence:

51h
A9h
03h

OUTD

Operation

(C) ← (HL), B ← B – 1, HL ← HL – 1

Op Code

OUTD

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	1	1	AB

Operands

None.

Description

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B can be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next, the byte to be output is placed on the data bus and written to the selected peripheral device. Finally, the register pair HL is decremented.

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3. 4)	4.00

Condition Bits Affected

S is unknown.

Z is set if B – 1 = 0; otherwise, it is reset.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

If Register C contains `07h`, Register B contains `10h`, the HL register pair contains `1000h`, and memory location `1000h` contains `59h`, then upon the execution of an OUTD instruction, Register B contains `0Fh`, the HL register pair contains `0FFFh`, and byte `59h` is written to the peripheral device mapped to I/O port address `07h`.

OTDR

Operation

(C) ← (HL), B ← B – 1, HL ← HL – 1

Op Code

OTDR

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	1	1	BB

Operands

None.

Description

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of Register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B can be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next, the byte to be output is placed on the data bus and written to the selected peripheral device. Then, register pair HL is decremented and if the decremented B Register is not 0, the Program Counter (PC) is decremented by two and the instruction is repeated. If B has gone to 0, the instruction is terminated. Interrupts are recognized and two refresh cycles are executed after each data transfer.

Note: When B is set to 0 prior to instruction execution, the instruction outputs 256 bytes of data.

If B ≠ 0:

M Cycles	T States	4 MHz E.T.
5	21 (4, 5, 3, 4, 5)	5.25

If B = 0:

M Cycles	T States	4 MHz E.T.
4	16 (4, 5, 3, 4)	4.00

Condition Bits Affected

S is unknown.

Z is set.

H is unknown.

P/V is unknown.

N is set.

C is not affected.

Example

Register C contains `07h`, Register B contains `03h`, the HL register pair contains `1000h`, and memory locations contain the following data.

0FFEh	51h
0FFFh	A9h
1000h	03h

Upon the execution of an OTDR instruction, the HL register pair contain `0FFDh`, Register B contains a 0, and a group of bytes is written to the peripheral device mapped to I/O port address `07h` in the following sequence:

03h
A9h
51h

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with ou1r products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.