

CHAPTER 4 INSTRUCTION SET

A computer, no matter how sophisticated, can only do what it is “told” to do. One “tells” the computer what to do via a series of coded instructions referred to as a **Program**. The realm of the programmer is referred to as **Software**, in contrast to the **Hardware** that comprises the actual computer equipment. A computer’s software refers to all of the programs that have been written for that computer.

When a computer is designed, the engineers provide the Central Processing Unit (CPU) with the ability to perform a particular set of operations. The CPU is designed such that a specific operation is performed when the CPU control logic decodes a particular instruction. Consequently, the operations that can be performed by a CPU define the computer’s **Instruction Set**.

Each computer instruction allows the programmer to initiate the performance of a specific operation. All computers implement certain arithmetic operations in their instruction set, such as an instruction to add the contents of two registers. Often logical operations (e.g., OR the contents of two registers) and register operate instructions (e.g., increment a register) are included in the instruction set. A computer’s instruction set will also have instructions that move data between registers, between a register and memory, and between a register and an I/O device. Most instruction sets also provide **Conditional Instructions**. A conditional instruction specifies an operation to be performed only if certain conditions have been met; for example, jump to a particular instruction if the result of the last operation was zero. Conditional instructions provide a program with a decision-making capability.

By logically organizing a sequence of instructions into a coherent program, the programmer can “tell” the computer to perform a very specific and useful function.

The computer, however, can only execute programs whose instructions are in a binary coded form (i.e., a series of 1’s and 0’s), that is called **Machine Code**. Because it would be extremely cumbersome to program in machine code, programming languages have been developed. There

are programs available which convert the programming language instructions into machine code that can be interpreted by the processor.

One type of programming language is **Assembly Language**. A unique assembly language mnemonic is assigned to each of the computer’s instructions. The programmer can write a program (called the **Source Program**) using these mnemonics and certain operands; the source program is then converted into machine instructions (called the **Object Code**). Each assembly language instruction is converted into one machine code instruction (1 or more bytes) by an **Assembler** program. Assembly languages are usually machine dependent (i.e., they are usually able to run on only one type of computer).

THE 8080 INSTRUCTION SET

The 8080 instruction set includes five different types of instructions:

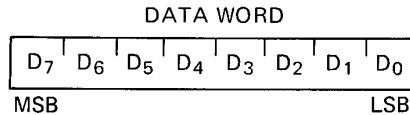
- **Data Transfer Group**—move data between registers or between memory and registers
- **Arithmetic Group** — add, subtract, increment or decrement data in registers or in memory
- **Logical Group** — AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory
- **Branch Group** — conditional and unconditional jump instructions, subroutine call instructions and return instructions
- **Stack, I/O and Machine Control Group** — includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

Instruction and Data Formats:

Memory for the 8080 is organized into 8-bit quantities, called Bytes. Each byte has a unique 16-bit binary address corresponding to its sequential position in memory.

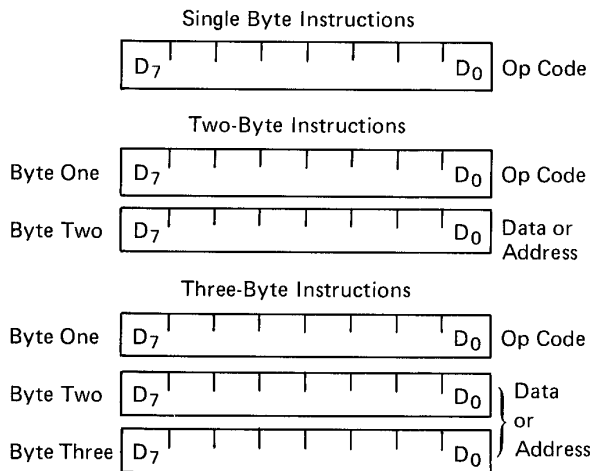
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:



When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the **Least Significant Bit (LSB)**, and BIT 7 (of an 8 bit number) is referred to as the **Most Significant Bit (MSB)**.

The 8080 program instructions may be one, two or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



Addressing Modes:

Often the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations, with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- **Direct** — Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- **Register** — The instruction specifies the register or register-pair in which the data is located.
- **Register Indirect** — The instruction specifies a register-pair which contains the memory

address where the data is located (the high-order bits of the address are in the first register of the pair, the low-order bits in the second).

- **Immediate** — The instruction contains the data itself. This is either an 8-bit quantity or a 16-bit quantity (least significant byte first, most significant byte second).

Unless directed by an interrupt or branch instruction, the execution of instructions proceeds through consecutively increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- **Direct** — The branch instruction contains the address of the next instruction to be executed. (Except for the 'RST' instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- **Register indirect** — The branch instruction indicates a register-pair which contains the address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RST instruction is a special one-byte call instruction (usually used during interrupt sequences). RST includes a three-bit field; program control is transferred to the instruction whose address is eight times the contents of this three-bit field.

Condition Flags:

There are five condition flags associated with the execution of instructions on the 8080. They are Zero, Sign, Parity, Carry, and Auxiliary Carry, and are each represented by a 1-bit register in the CPU. A flag is "set" by forcing the bit to 1; "reset" by forcing the bit to 0.

Unless indicated otherwise, when an instruction affects a flag, it affects it in the following manner:

- Zero:** If the result of an instruction has the value 0, this flag is set; otherwise it is reset.
- Sign:** If the most significant bit of the result of the operation has the value 1, this flag is set; otherwise it is reset.
- Parity:** If the modulo 2 sum of the bits of the result of the operation is 0, (i.e., if the result has even parity), this flag is set; otherwise it is reset (i.e., if the result has odd parity).
- Carry:** If the instruction resulted in a carry (from addition), or a borrow (from subtraction or a comparison) out of the high-order bit, this flag is set; otherwise it is reset.

Auxiliary Carry: If the instruction caused a carry out of bit 3 and into bit 4 of the resulting value, the auxiliary carry is set; otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments preceding a DAA (Decimal Adjust Accumulator) instruction.

Symbols and Abbreviations:

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

SYMBOLS MEANING

accumulator	Register A
addr	16-bit address quantity
data	8-bit data quantity
data 16	16-bit data quantity
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
port	8-bit address of an I/O device
r,r1,r2	One of the registers A,B,C,D,E,H,L
DDD,SSS	The bit pattern designating one of the registers A,B,C,D,E,H,L (DDD=destination, SSS=source):

DDD or SSS REGISTER NAME

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

rp	One of the register pairs: B represents the B,C pair with B as the high-order register and C as the low-order register; D represents the D,E pair with D as the high-order register and E as the low-order register; H represents the H,L pair with H as the high-order register and L as the low-order register; SP represents the 16-bit stack pointer register.
RP	The bit pattern designating one of the register pairs B,D,H,SP:

RP	REGISTER PAIR
00	B-C
01	D-E
10	H-L
11	SP

rh	The first (high-order) register of a designated register pair.
rl	The second (low-order) register of a designated register pair.
PC	16-bit program counter register (PCH and PCL are used to refer to the high-order and low-order 8 bits respectively).
SP	16-bit stack pointer register (SPH and SPL are used to refer to the high-order and low-order 8 bits respectively).
r _m	Bit m of the register r (bits are number 7 through 0 from left to right).
Z,S,P,CY,AC	The condition flags: Zero, Sign, Parity, Carry, and Auxiliary Carry, respectively.
()	The contents of the memory location or registers enclosed in the parentheses.
←	"Is transferred to"
∧	Logical AND
⊕	Exclusive OR
∨	Inclusive OR
+	Addition
−	Two's complement subtraction
*	Multiplication
↔	"Is exchanged with"
—	The one's complement (e.g., (A̅))
n	The restart number 0 through 7
NNN	The binary representation 000 through 111 for restart number 0 through 7 respectively.

Description Format:

The following pages provide a detailed description of the instruction set of the 8080. Each instruction is described in the following manner:

1. The MAC 80 assembler format, consisting of the instruction mnemonic and operand fields, is printed in **BOLDFACE** on the left side of the first line.
2. The name of the instruction is enclosed in parenthesis on the right side of the first line.
3. The next line(s) contain a symbolic description of the operation of the instruction.
4. This is followed by a narrative description of the operation of the instruction.
5. The following line(s) contain the binary fields and patterns that comprise the machine instruction.

6. The last four lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times, as in a Conditional Jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see Page 4-2) are listed. The last line lists any of the five Flags that are affected by the execution of the instruction.

Data Transfer Group:

This group of instructions transfers data to and from registers and memory. **Condition flags are not affected** by any instruction in this group.

MOV r1, r2 (Move Register)

(r1) ← (r2)

The content of register r2 is moved to register r1.

0	1	D	D	D	S	S	S
---	---	---	---	---	---	---	---

Cycles: 1
States: 5
Addressing: register
Flags: none

MOV r, M (Move from memory)

(r) ← ((H) (L))

The content of the memory location, whose address is in registers H and L, is moved to register r.

0	1	D	D	D	1	1	0
---	---	---	---	---	---	---	---

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

MOV M, r (Move to memory)

((H) (L)) ← (r)

The content of register r is moved to the memory location whose address is in registers H and L.

0	1	1	1	0	S	S	S
---	---	---	---	---	---	---	---

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

MVI r, data (Move Immediate)

(r) ← (byte 2)

The content of byte 2 of the instruction is moved to register r.

0	0	D	D	D	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: none

MVI M, data (Move to memory immediate)

((H) (L)) ← (byte 2)

The content of byte 2 of the instruction is moved to the memory location whose address is in registers H and L.

0	0	1	1	0	1	1	0
data							

Cycles: 3
States: 10
Addressing: immed./reg. indirect
Flags: none

LXI rp, data 16 (Load register pair immediate)

(rh) ← (byte 3),

(rl) ← (byte 2)

Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.

0	0	R	P	0	0	0	1
low-order data							
high-order data							

Cycles: 3
States: 10
Addressing: immediate
Flags: none

LDA addr (Load Accumulator direct) $(A) \leftarrow ((\text{byte } 3)(\text{byte } 2))$

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.

0	0	1	1	1	0	1	0
low-order addr							
high-order addr							

Cycles: 4
 States: 13
 Addressing: direct
 Flags: none

STA addr (Store Accumulator direct) $((\text{byte } 3)(\text{byte } 2)) \leftarrow (A)$

The content of the accumulator is moved to the memory location whose address is specified in byte 2 and byte 3 of the instruction.

0	0	1	1	0	0	1	0
low-order addr							
high-order addr							

Cycles: 4
 States: 13
 Addressing: direct
 Flags: none

LHLD addr (Load H and L direct) $(L) \leftarrow ((\text{byte } 3)(\text{byte } 2))$ $(H) \leftarrow ((\text{byte } 3)(\text{byte } 2) + 1)$

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register L. The content of the memory location at the succeeding address is moved to register H.

0	0	1	0	1	0	1	0
low-order addr							
high-order addr							

Cycles: 5
 States: 16
 Addressing: direct
 Flags: none

SHLD addr (Store H and L direct) $((\text{byte } 3)(\text{byte } 2)) \leftarrow (L)$ $((\text{byte } 3)(\text{byte } 2) + 1) \leftarrow (H)$

The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.

0	0	1	0	0	0	1	0
low-order addr							
high-order addr							

Cycles: 5
 States: 16
 Addressing: direct
 Flags: none

LDAX rp (Load accumulator indirect) $(A) \leftarrow ((rp))$

The content of the memory location, whose address is in the register pair rp, is moved to register A. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.

0	0	R	P	1	0	1	0
---	---	---	---	---	---	---	---

Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: none

STAX rp (Store accumulator indirect) $((rp)) \leftarrow (A)$

The content of register A is moved to the memory location whose address is in the register pair rp. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.

0	0	R	P	0	0	1	0
---	---	---	---	---	---	---	---

Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: none

XCHG (Exchange H and L with D and E) $(H) \leftrightarrow (D)$ $(L) \leftrightarrow (E)$

The contents of registers H and L are exchanged with the contents of registers D and E.

1	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---

Cycles: 1
 States: 4
 Addressing: register
 Flags: none

Arithmetic Group:

This group of instructions performs arithmetic operations on data in registers and memory.

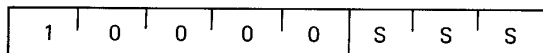
Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags according to the standard rules.

All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

ADD r (Add Register)

$$(A) \leftarrow (A) + (r)$$

The content of register r is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

States: 4

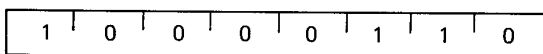
Addressing: register

Flags: Z,S,P,CY,AC

ADD M (Add memory)

$$(A) \leftarrow (A) + ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

States: 7

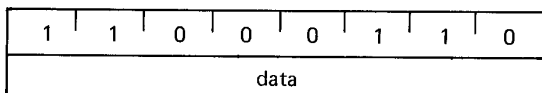
Addressing: reg. indirect

Flags: Z,S,P,CY,AC

ADI data (Add immediate)

$$(A) \leftarrow (A) + (\text{byte 2})$$

The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

States: 7

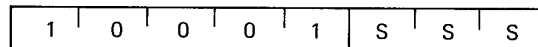
Addressing: immediate

Flags: Z,S,P,CY,AC

ADC r (Add Register with carry)

$$(A) \leftarrow (A) + (r) + (CY)$$

The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

States: 4

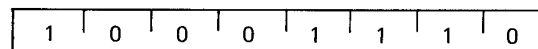
Addressing: register

Flags: Z,S,P,CY,AC

ADC M (Add memory with carry)

$$(A) \leftarrow (A) + ((H) (L)) + (CY)$$

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.



Cycles: 2

States: 7

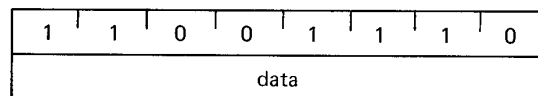
Addressing: reg. indirect

Flags: Z,S,P,CY,AC

ACI data (Add immediate with carry)

$$(A) \leftarrow (A) + (\text{byte 2}) + (CY)$$

The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.



Cycles: 2

States: 7

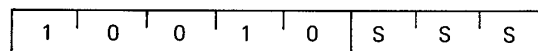
Addressing: immediate

Flags: Z,S,P,CY,AC

SUB r (Subtract Register)

$$(A) \leftarrow (A) - (r)$$

The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

States: 4

Addressing: register

Flags: Z,S,P,CY,AC

SUB M (Subtract memory)

$$(A) \leftarrow (A) - ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	0	0	1	0	1	1	0
---	---	---	---	---	---	---	---

Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: Z,S,P,CY,AC

SUI data (Subtract immediate)

$$(A) \leftarrow (A) - (\text{byte 2})$$

The content of the second byte of the instruction is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	1	0	1	0	1	1	0
data							

Cycles: 2
 States: 7
 Addressing: immediate
 Flags: Z,S,P,CY,AC

SBB r (Subtract Register with borrow)

$$(A) \leftarrow (A) - (r) - (CY)$$

The content of register r and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	S	S	S
---	---	---	---	---	---	---	---

Cycles: 1
 States: 4
 Addressing: register
 Flags: Z,S,P,CY,AC

SBB M (Subtract memory with borrow)

$$(A) \leftarrow (A) - ((H) (L)) - (CY)$$

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	1	1	0
---	---	---	---	---	---	---	---

Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: Z,S,P,CY,AC

SBI data (Subtract immediate with borrow)

$$(A) \leftarrow (A) - (\text{byte 2}) - (CY)$$

The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	1	0	1	1	1	1	0
data							

Cycles: 2
 States: 7
 Addressing: immediate
 Flags: Z,S,P,CY,AC

INR r (Increment Register)

$$(r) \leftarrow (r) + 1$$

The content of register r is incremented by one. Note: All condition flags **except** CY are affected.

0	0	D	D	D	1	0	0
---	---	---	---	---	---	---	---

Cycles: 1
 States: 5
 Addressing: register
 Flags: Z,S,P,AC

INR M (Increment memory)

$$((H) (L)) \leftarrow ((H) (L)) + 1$$

The content of the memory location whose address is contained in the H and L registers is incremented by one. Note: All condition flags **except** CY are affected.

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Cycles: 3
 States: 10
 Addressing: reg. indirect
 Flags: Z,S,P,AC

DCR r (Decrement Register)

$$(r) \leftarrow (r) - 1$$

The content of register r is decremented by one. Note: All condition flags **except** CY are affected.

0	0	D	D	D	1	0	1
---	---	---	---	---	---	---	---

Cycles: 1
 States: 5
 Addressing: register
 Flags: Z,S,P,AC

DCR M (Decrement memory)
$$((H) (L)) \leftarrow ((H) (L)) - 1$$

The content of the memory location whose address is contained in the H and L registers is decremented by one. Note: All condition flags **except CY** are affected.

0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 10
 Addressing: reg. indirect
 Flags: Z,S,P,AC

INX rp (Increment register pair)
$$(rh) (rl) \leftarrow (rh) (rl) + 1$$

The content of the register pair *rp* is incremented by one. Note: **No condition flags are affected.**

0	0	R	P	0	0	1	1
---	---	---	---	---	---	---	---

Cycles: 1
 States: 5
 Addressing: register
 Flags: none

DCX rp (Decrement register pair)
$$(rh) (rl) \leftarrow (rh) (rl) - 1$$

The content of the register pair *rp* is decremented by one. Note: **No condition flags are affected.**

0	0	R	P	1	0	1	1
---	---	---	---	---	---	---	---

Cycles: 1
 States: 5
 Addressing: register
 Flags: none

DAD rp (Add register pair to H and L)
$$(H) (L) \leftarrow (H) (L) + (rh) (rl)$$

The content of the register pair *rp* is added to the content of the register pair H and L. The result is placed in the register pair H and L. Note: **Only the CY flag is affected.** It is set if there is a carry out of the double precision add; otherwise it is reset.

0	0	R	P	1	0	0	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 10
 Addressing: register
 Flags: CY

DAA (Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two four-bit Binary-Coded-Decimal digits by the following process:

1. If the value of the least significant 4 bits of the accumulator is greater than 9 or if the AC flag is set, 6 is added to the accumulator.
2. If the value of the most significant 4 bits of the accumulator is now greater than 9, or if the CY flag is set, 6 is added to the most significant 4 bits of the accumulator.

NOTE: All flags are affected.

0	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---

Cycles: 1
 States: 4
 Flags: Z,S,P,CY,AC

Logical Group:

This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

ANA r (AND Register)
$$(A) \leftarrow (A) \wedge (r)$$

The content of register *r* is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

1	0	1	0	0	S	S	S
---	---	---	---	---	---	---	---

Cycles: 1
 States: 4
 Addressing: register
 Flags: Z,S,P,CY,AC

ANA M (AND memory)
$$(A) \leftarrow (A) \wedge ((H) (L))$$

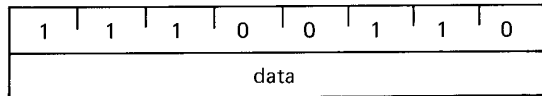
The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

1	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: Z,S,P,CY,AC

ANI data (AND immediate) $(A) \leftarrow (A) \wedge (\text{byte } 2)$

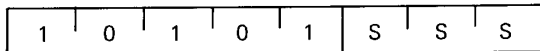
The content of the second byte of the instruction is logically anded with the contents of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2
 States: 7
 Addressing: immediate
 Flags: Z,S,P,CY,AC

XRA r (Exclusive OR Register) $(A) \leftarrow (A) \vee (r)$

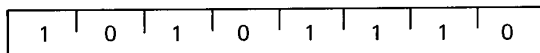
The content of register *r* is exclusive-or'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 1
 States: 4
 Addressing: register
 Flags: Z,S,P,CY,AC

XRA M (Exclusive OR Memory) $(A) \leftarrow (A) \vee ((H) (L))$

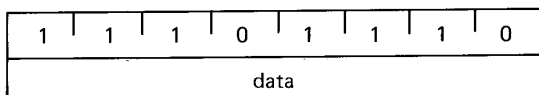
The content of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: Z,S,P,CY,AC

XRI data (Exclusive OR immediate) $(A) \leftarrow (A) \vee (\text{byte } 2)$

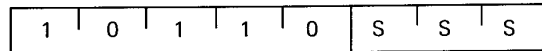
The content of the second byte of the instruction is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2
 States: 7
 Addressing: immediate
 Flags: Z,S,P,CY,AC

ORA r (OR Register) $(A) \leftarrow (A) \vee (r)$

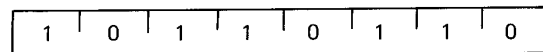
The content of register *r* is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 1
 States: 4
 Addressing: register
 Flags: Z,S,P,CY,AC

ORA M (OR memory) $(A) \leftarrow (A) \vee ((H) (L))$

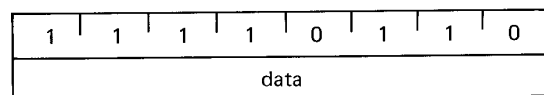
The content of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: Z,S,P,CY,AC

ORI data (OR Immediate) $(A) \leftarrow (A) \vee (\text{byte } 2)$

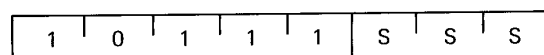
The content of the second byte of the instruction is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2
 States: 7
 Addressing: immediate
 Flags: Z,S,P,CY,AC

CMP r (Compare Register) $(A) - (r)$

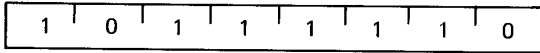
The content of register *r* is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. **The Z flag is set to 1 if $(A) = (r)$. The CY flag is set to 1 if $(A) < (r)$.**



Cycles: 1
 States: 4
 Addressing: register
 Flags: Z,S,P,CY,AC

CMP M (Compare memory) $(A) - ((H) (L))$

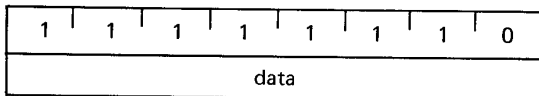
The content of the memory location whose address is contained in the H and L registers is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if $(A) = ((H) (L))$. The CY flag is set to 1 if $(A) < ((H) (L))$.



Cycles: 2
 States: 7
 Addressing: reg. indirect
 Flags: Z,S,P,CY,AC

CPI data (Compare immediate) $(A) - (\text{byte } 2)$

The content of the second byte of the instruction is subtracted from the accumulator. The condition flags are set by the result of the subtraction. The Z flag is set to 1 if $(A) = (\text{byte } 2)$. The CY flag is set to 1 if $(A) < (\text{byte } 2)$.

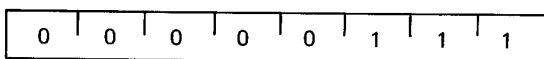


Cycles: 2
 States: 7
 Addressing: immediate
 Flags: Z,S,P,CY,AC

RLC (Rotate left)
$$(A_{n+1}) \leftarrow (A_n) ; (A_0) \leftarrow (A_7)$$

$$(CY) \leftarrow (A_7)$$

The content of the accumulator is rotated left one position. The low order bit and the CY flag are both set to the value shifted out of the high order bit position. **Only the CY flag is affected.**

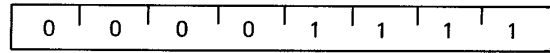


Cycles: 1
 States: 4
 Flags: CY

RRC (Rotate right)
$$(A_n) \leftarrow (A_{n-1}) ; (A_7) \leftarrow (A_0)$$

$$(CY) \leftarrow (A_0)$$

The content of the accumulator is rotated right one position. The high order bit and the CY flag are both set to the value shifted out of the low order bit position. **Only the CY flag is affected.**

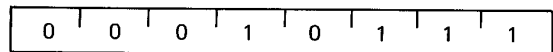


Cycles: 1
 States: 4
 Flags: CY

RAL (Rotate left through carry)
$$(A_{n+1}) \leftarrow (A_n) ; (CY) \leftarrow (A_7)$$

$$(A_0) \leftarrow (CY)$$

The content of the accumulator is rotated left one position through the CY flag. The low order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high order bit. **Only the CY flag is affected.**

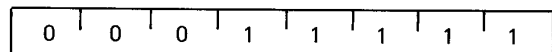


Cycles: 1
 States: 4
 Flags: CY

RAR (Rotate right through carry)
$$(A_n) \leftarrow (A_{n+1}) ; (CY) \leftarrow (A_0)$$

$$(A_7) \leftarrow (CY)$$

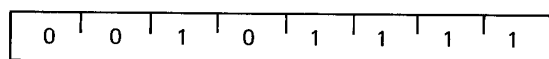
The content of the accumulator is rotated right one position through the CY flag. The high order bit is set to the CY flag and the CY flag is set to the value shifted out of the low order bit. **Only the CY flag is affected.**



Cycles: 1
 States: 4
 Flags: CY

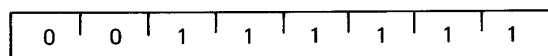
CMA (Complement accumulator) $(A) \leftarrow (\bar{A})$

The contents of the accumulator are complemented (zero bits become 1, one bits become 0). **No flags are affected.**



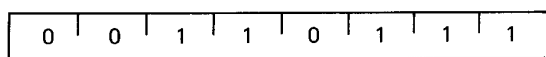
Cycles: 1
 States: 4
 Flags: none

CMC (Complement carry)
 $(CY) \leftarrow \overline{(CY)}$
 The CY flag is complemented. **No other flags are affected.**



Cycles: 1
 States: 4
 Flags: CY

STC (Set carry)
 $(CY) \leftarrow 1$
 The CY flag is set to 1. **No other flags are affected.**



Cycles: 1
 States: 4
 Flags: CY

Branch Group:

This group of instructions alter normal sequential program flow.

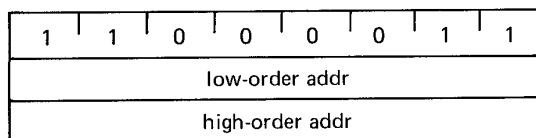
Condition flags are not affected by any instruction in this group.

The two types of branch instructions are unconditional and conditional. Unconditional transfers simply perform the specified operation on register PC (the program counter). Conditional transfers examine the status of one of the four processor flags to determine if the specified branch is to be executed. The conditions that may be specified are as follows:

CONDITION	CCC
NZ — not zero ($Z = 0$)	000
Z — zero ($Z = 1$)	001
NC — no carry ($CY = 0$)	010
C — carry ($CY = 1$)	011
PO — parity odd ($P = 0$)	100
PE — parity even ($P = 1$)	101
P — plus ($S = 0$)	110
M — minus ($S = 1$)	111

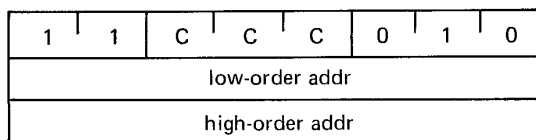
JMP addr (Jump)
 $(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$
 Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

address is specified in byte 3 and byte 2 of the current instruction.



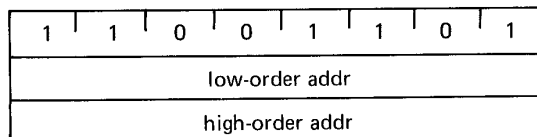
Cycles: 3
 States: 10
 Addressing: immediate
 Flags: none

Jcondition addr (Conditional jump)
 If (CCC),
 $(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$
 If the specified condition is true, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction; otherwise, control continues sequentially.



Cycles: 3
 States: 10
 Addressing: immediate
 Flags: none

CALL addr (Call)
 $((SP) - 1) \leftarrow (PCH)$
 $((SP) - 2) \leftarrow (PCL)$
 $(SP) \leftarrow (SP) - 2$
 $(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$
 The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.



Cycles: 5
 States: 17
 Addressing: immediate/reg. indirect
 Flags: none

Ccondition addr (Condition call)

If (CCC),
 $((SP) - 1) \leftarrow (PCH)$
 $((SP) - 2) \leftarrow (PCL)$
 $(SP) \leftarrow (SP) - 2$
 $(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$

If the specified condition is true, the actions specified in the CALL instruction (see above) are performed; otherwise, control continues sequentially.

1	1	C	C	C	1	0	0
low-order addr							
high-order addr							

Cycles: 3/5
 States: 11/17
 Addressing: immediate/reg. indirect
 Flags: none

RET (Return)

$(PCL) \leftarrow ((SP));$
 $(PCH) \leftarrow ((SP) + 1);$
 $(SP) \leftarrow (SP) + 2;$

The content of the memory location whose address is specified in register SP is moved to the low-order eight bits of register PC. The content of the memory location whose address is one more than the content of register SP is moved to the high-order eight bits of register PC. The content of register SP is incremented by 2.

1	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 10
 Addressing: reg. indirect
 Flags: none

Rcondition (Conditional return)

If (CCC),
 $(PCL) \leftarrow ((SP))$
 $(PCH) \leftarrow ((SP) + 1)$
 $(SP) \leftarrow (SP) + 2$

If the specified condition is true, the actions specified in the RET instruction (see above) are performed; otherwise, control continues sequentially.

1	1	C	C	C	0	0	0
---	---	---	---	---	---	---	---

Cycles: 1/3
 States: 5/11
 Addressing: reg. indirect
 Flags: none

RST n (Restart)

$((SP) - 1) \leftarrow (PCH)$
 $((SP) - 2) \leftarrow (PCL)$
 $(SP) \leftarrow (SP) - 2$
 $(PC) \leftarrow 8 * (NNN)$

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two. Control is transferred to the instruction whose address is eight times the content of NNN.

1	1	N	N	N	1	1	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 11
 Addressing: reg. indirect
 Flags: none

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	N	N	N	0	0	0

Program Counter After Restart

PCHL (Jump H and L indirect — move H and L to PC)

$(PCH) \leftarrow (H)$
 $(PCL) \leftarrow (L)$

The content of register H is moved to the high-order eight bits of register PC. The content of register L is moved to the low-order eight bits of register PC.

1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

Cycles: 1
 States: 5
 Addressing: register
 Flags: none

Stack, I/O, and Machine Control Group:

This group of instructions performs I/O, manipulates the Stack, and alters internal control flags.

Unless otherwise specified, **condition flags are not affected by any instructions in this group.**

PUSH rp (Push)

$((SP) - 1) \leftarrow (rh)$
 $((SP) - 2) \leftarrow (rl)$
 $(SP) \leftarrow (SP) - 2$

The content of the high-order register of register pair rp is moved to the memory location whose address is one less than the content of register SP. The content of the low-order register of register pair rp is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. **Note: Register pair rp = SP may not be specified.**

1	1	R	P	0	1	0	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 11
 Addressing: reg. indirect
 Flags: none

PUSH PSW (Push processor status word)

$((SP) - 1) \leftarrow (A)$
 $((SP) - 2)_0 \leftarrow (CY), ((SP) - 2)_1 \leftarrow 1$
 $((SP) - 2)_2 \leftarrow (P), ((SP) - 2)_3 \leftarrow 0$
 $((SP) - 2)_4 \leftarrow (AC), ((SP) - 2)_5 \leftarrow 0$
 $((SP) - 2)_6 \leftarrow (Z), ((SP) - 2)_7 \leftarrow (S)$
 $(SP) \leftarrow (SP) - 2$

The content of register A is moved to the memory location whose address is one less than register SP. The contents of the condition flags are assembled into a processor status word and the word is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two.

1	1	1	1	0	1	0	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 11
 Addressing: reg. indirect
 Flags: none

FLAG WORD

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	0	AC	0	P	1	CY

POP rp (Pop)

$(rl) \leftarrow ((SP))$
 $(rh) \leftarrow ((SP) + 1)$
 $(SP) \leftarrow (SP) + 2$

The content of the memory location, whose address is specified by the content of register SP, is moved to the low-order register of register pair rp. The content of the memory location, whose address is one more than the content of register SP, is moved to the high-order register of register pair rp. The content of register SP is incremented by 2. **Note: Register pair rp = SP may not be specified.**

1	1	R	P	0	0	0	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 10
 Addressing: reg. indirect
 Flags: none

POP PSW (Pop processor status word)

$(CY) \leftarrow ((SP))_0$
 $(P) \leftarrow ((SP))_2$
 $(AC) \leftarrow ((SP))_4$
 $(Z) \leftarrow ((SP))_6$
 $(S) \leftarrow ((SP))_7$
 $(A) \leftarrow ((SP) + 1)$
 $(SP) \leftarrow (SP) + 2$

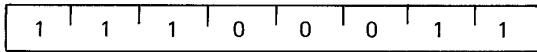
The content of the memory location whose address is specified by the content of register SP is used to restore the condition flags. The content of the memory location whose address is one more than the content of register SP is moved to register A. The content of register SP is incremented by 2.

1	1	1	1	0	0	0	1
---	---	---	---	---	---	---	---

Cycles: 3
 States: 10
 Addressing: reg. indirect
 Flags: Z,S,P,CY,AC

XTHL (Exchange stack top with H and L)(L) \leftrightarrow ((SP))(H) \leftrightarrow ((SP) + 1)

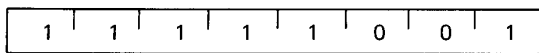
The content of the L register is exchanged with the content of the memory location whose address is specified by the content of register SP. The content of the H register is exchanged with the content of the memory location whose address is one more than the content of register SP.



Cycles: 5
 States: 18
 Addressing: reg. indirect
 Flags: none

SPHL (Move HL to SP)(SP) \leftarrow (H) (L)

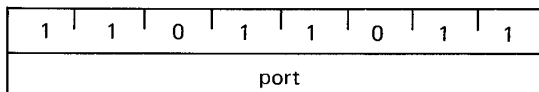
The contents of registers H and L (16 bits) are moved to register SP.



Cycles: 1
 States: 5
 Addressing: register
 Flags: none

IN port (Input)(A) \leftarrow (data)

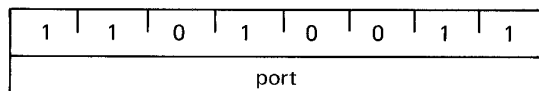
The data placed on the eight bit bi-directional data bus by the specified port is moved to register A.



Cycles: 3
 States: 10
 Addressing: direct
 Flags: none

OUT port (Output)(data) \leftarrow (A)

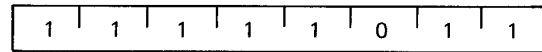
The content of register A is placed on the eight bit bi-directional data bus for transmission to the specified port.



Cycles: 3
 States: 10
 Addressing: direct
 Flags: none

EI (Enable interrupts)

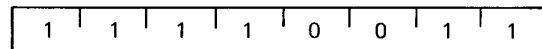
The interrupt system is enabled **following the execution of the next instruction.**



Cycles: 1
 States: 4
 Flags: none

DI (Disable interrupts)

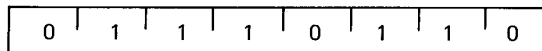
The interrupt system is disabled **immediately following the execution of the DI instruction.**



Cycles: 1
 States: 4
 Flags: none

HLT (Halt)

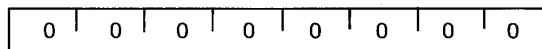
The processor is stopped. The registers and flags are unaffected.



Cycles: 1
 States: 7
 Flags: none

NOP (No op)

No operation is performed. The registers and flags are unaffected.



Cycles: 1
 States: 4
 Flags: none

INSTRUCTION SET

Summary of Processor Instructions

Mnemonic	Description	Instruction Code ^[1]								Clock ^[2] Cycles
		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
MOV _{r1,r2}	Move register to register	0	1	0	0	0	S	S	S	5
MOV M, r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r, M	Move memory to register	0	1	0	0	0	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	0	0	0	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	0	0	0	1	0	0	5
DCR r	Decrement register	0	0	0	0	0	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	0	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	0	0	1	1	4
DI	Disable Interrupts	1	1	1	1	0	0	1	1	4
NOP	No-operation	0	0	0	0	0	0	0	0	4

NOTES: 1. DDD or SSS — 000 B — 001 C — 010 D — 011 E — 100 H — 101 L — 110 Memory — 111 A.
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.