CHAPTER 5
THE INSTRUCTION SET

3.1 WHAT THE INSTRUCTION SET IS

A computer, no matter how sophisticated, can do only what it is instructed to do. A program is a sequence of instructions, each of which is recognized by the computer and causes it to perform an operation. Once a program is placed in memory space that is accessible to your CPU, you may run that same sequence of instructions as often as you wish to solve the same problem or to do the same function. The set of instructions to which the 8085A CPU will respond is permanently fixed in the design of the chip.

Each computer instruction allows you to initiate the performance of a specific operation. The 8085A implements a group of instructions that move data between registers, between a register and memory, and between a register and an I/O port. It also has arithmetic and logic instructions, conditional and unconditional branch instructions, and machine control instructions. The CPU recognizes these instructions only when they are coded in binary form.

3.2 SYMBOLS AND ABBREVIATIONS:

The following symbols and abbreviations are used in the subsequent description of the 8085A instructions:

<table>
<thead>
<tr>
<th>SYMBOLS</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>accumulator</td>
<td>Register A</td>
</tr>
<tr>
<td>addr</td>
<td>16-bit address quantity</td>
</tr>
<tr>
<td>data</td>
<td>8-bit quantity</td>
</tr>
<tr>
<td>data 16</td>
<td>16-bit data quantity</td>
</tr>
<tr>
<td>byte 2</td>
<td>The second byte of the instruction</td>
</tr>
<tr>
<td>byte 3</td>
<td>The third byte of the instruction</td>
</tr>
<tr>
<td>port</td>
<td>8-bit address of an I/O device</td>
</tr>
<tr>
<td>r,r1,r2</td>
<td>One of the registers A,B,C,D,E,H,L</td>
</tr>
</tbody>
</table>

The bit pattern designating one of the registers A,B,C,D,E,H,L

<table>
<thead>
<tr>
<th>DDD or SSS</th>
<th>REGISTER NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>A</td>
</tr>
<tr>
<td>000</td>
<td>B</td>
</tr>
<tr>
<td>001</td>
<td>C</td>
</tr>
<tr>
<td>010</td>
<td>D</td>
</tr>
<tr>
<td>011</td>
<td>E</td>
</tr>
<tr>
<td>100</td>
<td>H</td>
</tr>
<tr>
<td>101</td>
<td>L</td>
</tr>
</tbody>
</table>

The bit pattern designating 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.

The first (high-order) register of a designated register pair.

The second (low-order) register of a designated register pair.
5. The boxes describe the binary codes that comprise the machine instruction.

6. The last four lines contain 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 are listed, separated by a slash. Next, data addressing modes are listed if applicable. The last line lists any of the five flags that are affected by the execution of the instruction.

5.3 INSTRUCTION AND DATA FORMATS

Memory used in the MCS-85 system is organized in 8-bit bytes. Each byte has a unique location in physical memory. That location is described by one of a sequence of 16-bit binary addresses. The 8085A can address up to 64K (1024, or $2^{16}$; hence, 64K represents the decimal number 65,536) bytes of memory, which may consist of both random-access, read-write memory (RAM) and read-only memory (ROM), which is also random-access.

Data in the 8085A is stored in the form of 8-bit binary integers:

```
DATA WORD

<table>
<thead>
<tr>
<th>D_7</th>
<th>D_6</th>
<th>D_5</th>
<th>D_4</th>
<th>D_3</th>
<th>D_2</th>
<th>D_1</th>
<th>D_0</th>
</tr>
</thead>
</table>
```

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 8085A, 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)**.

An 8085A program instruction 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 instruction. The exact instruction format will depend on the particular operation to be executed.

**Single Byte Instructions**

```
| D_7 | D_6 | D_5 | D_4 | D_3 | D_2 | D_1 | D_0 | Op Code |
```

**Two-Byte Instructions**

```
Byte One | D_7 | D_6 | D_5 | D_4 | D_3 | D_2 | D_1 | D_0 | Op Code |

Byte Two | D_7 | D_6 | D_5 | D_4 | D_3 | D_2 | D_1 | D_0 | Data or Address |
```
5.4 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 8085A 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 whose address 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.

5.5 CONDITION FLAGS:

There are five condition flags associated with the execution of instructions on the 8085A. They are Zero, Sign, Parity, Carry, and Auxiliary Carry. Each is represented by a 1-bit register (or flip-flop) in the CPU. A flag is set by forcing the bit to 1; it is 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.
5.6 INSTRUCTION SET ENCYCLOPEDIA

In the ensuing dozen pages, the complete 8085A instruction set is described, grouped in order under five different functional headings, as follows:

1. **Data Transfer Group** — Moves data between registers or between memory locations and registers. Includes moves, loads, stores, and exchanges. (See below.)

2. **Arithmetic Group** — Adds, subtracts, increments, or decrements data in registers or memory. (See page 5-13.)

3. **Logic Group** — ANDs, ORs, XORs, compares, rotates, or complements data in registers or between memory and a register. (See page 5-16.)

4. **Branch Group** — Initiates conditional or unconditional jumps, calls, returns, and restarts. (See page 5-20.)

5. **Stack, I/O, and Machine Control Group** — Includes instructions for maintaining the stack, reading from input ports, writing to output ports, setting and reading interrupt masks, and setting and clearing flags. (See page 5-22.)

The formats described in the encyclopedia reflect the assembly language processed by Intel-supplied assembler, used with the Intellic® development systems.

5.6.1 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.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>S</th>
<th>S</th>
<th>S</th>
</tr>
</thead>
</table>

Cycles: 1  
States: 4 (8085), 5 (8080)  
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.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>S</th>
<th>S</th>
<th>S</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

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

*All mnemonics copyrighted ©Intel Corporation 1976.
**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.

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

---

**LHLD addr (Load H and L direct)**

(L) = (byte 3)(byte 2)
(H) = (byte 3)(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.

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

---

**LDA addr (Load Accumulator direct)**

(A) = (byte 3)(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.

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

---

**SHLD addr (Store H and L direct)**

((byte 3)(byte 2)) → (L)
((byte 3)(byte 2) + 1) → (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.

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

---

**STA addr (Store Accumulator direct)**

((byte 3)(byte 2)) → (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.

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

---

**LDAX rp (Load accumulator indirect)**

(A) = ([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.

Cycles: 2
States: 7
Addressing: reg. Indirect
Flags: none
THE INSTRUCTION SET

STAX rp  (Store accumulator indirect)
      ((rp)) ← (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.

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>R</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

XCHG  (Exchange H and L with D and E)
      (H) ← (D)
      (L) ← (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

5.6.2 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) ← (A) + (r)
The content of register r is added to the content of the accumulator. The result is placed in the accumulator.

| 1 | 0 | 0 | 0 | 0 | S | S | S |

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

ADD M  (Add memory)
      (A) ← (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.

| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

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

ADI data  (Add immediate)
      (A) ← (A) + (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.

| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |

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

ADC r  (Add Register with carry)
      (A) ← (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.

| 1 | 0 | 0 | 0 | 0 | S | S | S |

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

*All mnemonics copyrighted ©Intel Corporation 1976.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Cycles</th>
<th>States</th>
<th>Addressing</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADC M</strong></td>
<td>(Add memory with carry) ( (A) - (A) + ((H) \times (L)) + (CY) )</td>
<td>2</td>
<td>7</td>
<td>reg. indirect</td>
<td>Z,S,P,CY,AC</td>
</tr>
<tr>
<td></td>
<td>The content of the memory location whose address is contained in the H and L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>registers and the content of the CY flag are added to the accumulator. The</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>result is placed in the accumulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SUB M</strong></td>
<td>(Subtract memory) ( (A) - (A) - ((H) \times (L)) )</td>
<td>2</td>
<td>7</td>
<td>reg. indirect</td>
<td>Z,S,P,CY,AC</td>
</tr>
<tr>
<td></td>
<td>The content of the memory location whose address is contained in the H and L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>registers is subtracted from the content of the accumulator. The result is</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>placed in the accumulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ACI data</strong></td>
<td>(Add immediate with carry) ( (A) - (A) + (\text{byte 2}) + (CY) )</td>
<td>2</td>
<td>7</td>
<td>Immediate</td>
<td>Z,S,P,CY,AC</td>
</tr>
<tr>
<td></td>
<td>The content of the second byte of the instruction and the content of the CY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>flag are added to the contents of the accumulator. The result is placed in</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>the accumulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SUI data</strong></td>
<td>(Subtract immediate) ( (A) - (A) - (\text{byte 2}) )</td>
<td>2</td>
<td>7</td>
<td>Immediate</td>
<td>Z,S,P,CY,AC</td>
</tr>
<tr>
<td></td>
<td>The content of the second byte of the instruction is subtracted from the</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>contents of the accumulator. The result is placed in the accumulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SUB r</strong></td>
<td>(Subtract Register) ( (A) - (A) - (r) )</td>
<td>1</td>
<td>4</td>
<td>register</td>
<td>Z,S,P,CY,AC</td>
</tr>
<tr>
<td></td>
<td>The content of register ( r ) is subtracted from the content of the</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>accumulator. The result is placed in the accumulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SBB r</strong></td>
<td>(Subtract Register with borrow) ( (A) - (A) - (r) - (CY) )</td>
<td>1</td>
<td>4</td>
<td>register</td>
<td>Z,S,P,CY,AC</td>
</tr>
<tr>
<td></td>
<td>The content of register ( r ) and the content of the CY flag are both</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>subtracted from the accumulator. The result is placed in the accumulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**SBB M** (Subtract memory with borrow)

\[(A) - (A) - (((H) (L)) - (CY))\]

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.

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

**INR M** (Increment memory)

\[(((H) (L)) - (((H) (L)) + 1)\]

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.

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

**SBI data** (Subtract immediate with borrow)

\[(A) - (A) - (\text{byte 2}) - (CY)\]

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.

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

**DCR r** (Decrement Register)

\[(r) - (r) - 1\]

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

- **Cycles:** 1
- **States:** 4 (8085), 5 (8080)
- **Addressing:** register
- **Flags:** Z,S,P,AC

**INR r** (Increment Register)

\[(r) - (r) + 1\]

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

- **Cycles:** 1
- **States:** 4 (8085), 5 (8080)
- **Addressing:** register
- **Flags:** Z,S,P,AC

**DCR M** (Decrement memory)

\[(((H) (L)) - (((H) (L)) - 1)\]

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.

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

*All mnemonics copyrighted ©Intel Corporation 1976.*
INX rp  
(Increment register pair)
(rh) (rl) - (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: 6 (8085), 5 (8080)
- Addressing: register
- Flags: none

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 lease 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
```

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

DCX rp  
(Decrement register pair)
(rh) (rl) - (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: 6 (8085), 5 (8080)
- Addressing: register
- Flags: none

5.6.3 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.

DAD rp  
(Add register pair to H and L)
(H) (L) - (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

ANA r  
(AND Register)
(A) = (A) ∧ (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 and AC is set (8085). The CY flag is cleared and AC is set to the OR'ing of bits 3 of the operands (8080).

```
1 0 1 0 0 0 0 S S S S
```

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

*All mnemonics copyrighted © Intel Corporation 1976.*
ANA M (AND memory)
(A) ← (A) ∧ ((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 and AC is set (8085). The CY flag is cleared and AC is set to the OR’ing of bits 3 of the operands (8080).

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

XRA M (Exclusive OR Memory)
(A) ← (A) ∨ ((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

ANI data (AND immediate)
(A) ← (A) ∧ (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 flag is cleared and AC is set (8085). The CY flag is cleared and AC is set to the OR’ing of bits 3 of the operands (8080).

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

XRI data (Exclusive OR immediate)
(A) ← (A) ∨ (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

XRA r (Exclusive OR Register)
(A) ← (A) ∨ (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

ORA r (OR Register)
(A) ← (A) V (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)

\[(A) \rightarrow (A) \lor ((H) \land (L))\]

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.**

```
1 0 1 1 0 1 1 0
```

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

**ORI data**  (OR Immediate)

\[(A) \rightarrow (A) \lor (\text{byte 2})\]

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.**

```
1 1 1 1 0 1 1 0
```

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

**CMP M**  (Compare memory)

\[(A) \rightarrow ((H) \land (L))\]

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) \land (L)). The CY flag is set to 1 if (A) < ((H) \land (L)).**

```
1 0 1 1 1 1 1 0
```

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

**CPI data**  (Compare immediate)

\[(A) \rightarrow \text{(byte 2)}\]

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)}.**

```
1 1 1 1 1 1 1 0
```

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

**CMP r**  (Compare Register)

\[(A) \rightarrow (r)\]

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).**

```
1 0 1 1 1 1 1 1 1
```

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

**RLC**  (Rotate left)

\[(A_{n+1}) \rightarrow (A_0) ; (A_0) \rightarrow (A_7) ; (A_7) \rightarrow (A_7)\]

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.**

```
0 0 0 0 0 1 1 1 1
```

Cycles: 1  
States: 4  
Addressing:  
Flags: CY

*All mnemonics copyrighted © Intel Corporation 1976.*
THE INSTRUCTION SET

RRC  (Rotate right)
\((A_n) - (A_{n+1}); (A_7) - (A_0)\)
\((CY) - (A_7)\)
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.**

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}
\]

Cycles: 1
States: 4
Flags: CY

CMA  (Complement accumulator)
\((A) - (\overline{A})\)
The contents of the accumulator are complemented (zero bits become 1, one bits become 0). **No flags are affected.**

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

Cycles: 1
States: 4
Flags: none

RAL  (Rotate left through carry)
\((A_{n+1}) - (A_n); (CY) - (A_7)\)
\((A_0) - (CY)\)
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.**

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1
\end{array}
\]

Cycles: 1
States: 4
Flags: CY

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

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

Cycles: 1
States: 4
Flags: CY

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

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

Cycles: 1
States: 4
Flags: CY

RAR  (Rotate right through carry)
\((A_n) - (A_{n+1}); (CY) - (A_7)\)
\((A_0) - (CY)\)
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.**

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

Cycles: 1
States: 4
Flags: CY

*All mnemonics copyrighted © Intel Corporation 1976.*
5.6.4 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:

<table>
<thead>
<tr>
<th>CONDITION</th>
<th>CCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NZ — not zero (Z = 0)</td>
<td>000</td>
</tr>
<tr>
<td>Z — zero (Z = 1)</td>
<td>001</td>
</tr>
<tr>
<td>NC — no carry (CY = 0)</td>
<td>010</td>
</tr>
<tr>
<td>C — carry (CY = 1)</td>
<td>011</td>
</tr>
<tr>
<td>PO — parity odd (P = 0)</td>
<td>100</td>
</tr>
<tr>
<td>PE — parity even (P = 1)</td>
<td>101</td>
</tr>
<tr>
<td>P — plus (S = 0)</td>
<td>110</td>
</tr>
<tr>
<td>M — minus (S = 1)</td>
<td>111</td>
</tr>
</tbody>
</table>

JMP addr (Jump)

(PC) — (byte 3) (byte 2)

Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

Jcondition addr (Conditional jump)

If (CCC),

(PC) — (byte 3) (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.

<table>
<thead>
<tr>
<th>low-order addr</th>
<th>high-order addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cycles: 2/3 (8085), 3 (8080)
States: 7/10 (8085), 10 (8080)
Addressing: immediate
Flags: none

CALL addr (Call)

((SP) — 1) — (PCH)
((SP) — 2) — (PCL)
(SP) — (SP) — 2
(PC) — (byte 3) (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.

<table>
<thead>
<tr>
<th>low-order addr</th>
<th>high-order addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cycles: 5
States: 18 (8085), 17 (8080)
Addressing: reg. indirect
Flags: none

*All mnemonics copyrighted©Intel Corporation 1976.
Ccondition addr (Condition call)

If (CCC),

(SP) − 1) − (PCH)
(SP) − 2) − (PCL)
(SP) − (SP) − 2
(PC) − (byte 3) (byte 2)

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

Rcondition (Conditional return)

If (CCC),

(PCL) − ((SP))
(PCH) − ((SP) + 1)
(SP) − (SP) + 2

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

Cycles: 1/3
States: 6/12 (8085), 5/11 (8080)
Addressing: reg. indirect
Flags: none

RST n (Restart)

(SP) − 1) − (PCH)
(SP) − 2) − (PCL)
(SP) − (SP) − 2
(PC) − 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.

Cycles: 3
States: 12 (8085), 11 (8080)
Addressing: reg. indirect
Flags: none

Program Counter After Restart

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 0 0 0 0 0 NNN

All mnemonics copyrighted © Intel Corporation 1978.
**PCHL**  
(Jump H and L indirect — move H and L to PC)  
(PCHL) – (H)  
(PCL) – (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.  

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

Cycles: 1  
States: 6 (8085), 5 (8080)  
Addressing: register  
Flags: none

### 5.6.5 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) – (rh)  
((SP) – 2) – (rl)  
((SP) – (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.**

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>R</th>
<th>P</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

Cycles: 3  
States: 12 (8085), 11 (8080)  
Addressing: reg. indirect  
Flags: none

**POP rp**  
(Pop)  
(rl) – ((SP))  
(rh) – ((SP) + 1)  
((SP) – (SP) + 2)  
The content of the memory location, whose address is specified by the content of register SP, is moved to the lower-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 higher-order register of register rp. The content of register SP is incremented by 2. **Note: Register pair rp = SP may not be specified.**

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>R</th>
<th>P</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>

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

---

All mnemonics copyrighted © Intel Corporation 1976.
THE INSTRUCTION SET

**POP PSW**  (Pop processor status word)

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

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

**SPHL**  (Move HL to SP)

(SP) \( \rightarrow (H) (L) \)

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

```
1 1 1 1 1 0 0 1
```

Cycles: 1  
States: 6 (8085), 5 (8080)  
Addressing: register  
Flags: none

**IN port**  (Input)

(A) \( \rightarrow \) (data)

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

```
1 1 0 1 1 0 1 1
```

port

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

**OUT port**  (Output)

(data) \( \rightarrow \) (A)

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

```
1 1 0 1 0 0 1 1
```

port

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

*All mnemonics copyrighted ©Intel Corporation 1976.*
EI  
(Enable interrupts)
The interrupt system is enabled following the execution of the next instruction. Interrupts are not recognized during the EI instruction.

```
0 0 0 0 0 0 0 0
1 1 1 1 1 0 1 1
```
Cycles: 1
States: 4
Flags: none

NOTE: Placing an EI instruction on the bus in response to INTA during an INA cycle is prohibited. (8085)

DI  
(Disable interrupts)
The interrupt system is disabled immediately following the execution of the DI instruction. Interrupts are not recognized during the DI instruction.

```
0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 1
```
Cycles: 1
States: 4
Flags: none

NOTE: Placing a DI instruction on the bus in response to INTA during an INA cycle is prohibited. (8085)

HLT  
(Halt)
The processor is stopped. The registers and flags are unaffected. (8080) A second ALE is generated during the execution of HLT to strobe out the Halt cycle status information. (8085)

```
0 1 1 1 1 0 1 1 0
```
Cycles: 1 + (8085), 1 (8080)
States: 5 (8085), 7 (8080)
Flags: none

NOP  
(No op)
No operation is performed. The registers and flags are unaffected.

RIM  
(Read Interrupt Masks) (8085 only)
The RIM instruction loads data into the accumulator relating to interrupts and the serial input. This data contains the following information:

- Current interrupt mask status for the RST 5.5, 6.5, and 7.5 hardware interrupts (1 = mask disabled)
- Current interrupt enable flag status (1 = interrupts enabled) except immediately following a TRAP interrupt. (See below.)
- Hardware interrupts pending (i.e., signal received but not yet serviced), on the RST 5.5, 6.5, and 7.5 lines.
- Serial input data.

Immediately following a TRAP interrupt, the RIM instruction must be executed as a part of the service routine if you need to retrieve current interrupt status later. Bit 3 of the accumulator is (in this special case only) loaded with the interrupt enable (IE) flag status that existed prior to the TRAP interrupt. Following an RST 5.5, 6.5, 7.5, or INTR interrupt, the interrupt flag flip-flop reflects the current interrupt enable status. Bit 6 of the accumulator (I7.5) is loaded with the status of the RST 7.5 flip-flop, which is always set (edge-triggered) by an input on the RST 7.5 input line, even when that interrupt has been previously masked. (See SIM Instruction.)

*All mnemonics copyrighted © Intel Corporation 1976.*
SIM (Set Interrupt Masks) (8085 only)

The execution of the SIM instruction uses the contents of the accumulator (which must be previously loaded) to perform the following functions:

- Program the interrupt mask for the RST 5.5, 6.5, and 7.5 hardware interrupts.
- Reset the edge-triggered RST 7.5 input latch.
- Load the SOD output latch.

To program the interrupt masks, first set accumulator bit 3 to 1 and set to 1 any bits 0, 1, and 2, which disable interrupts RST 5.5, 6.5, and 7.5, respectively. Then do a SIM instruction. If accumulator bit 3 is 0 when the SIM instruction is executed, the interrupt mask register will not change. If accumulator bit 4 is 1 when the SIM instruction is executed, the RST 7.5 latch is then reset. RST 7.5 is distinguished by the fact that its latch is always set by a rising edge on the RST 7.5 input pin, even if the jump to service routine is inhibited by masking. This latch remains high until cleared by a RESET IN, by a SIM instruction with accumulator bit 4 high, or by an internal processor acknowledge to an RST 7.5 interrupt subsequent to the removal of the mask (by a SIM instruction). The RESET IN signal always sets all three RST mask bits.

If accumulator bit 6 is at the 1 level when the SIM instruction is executed, the state of accumulator bit 7 is loaded into the SOD latch and thus becomes available for interface to an external device. The SOD latch is unaffected by the SIM instruction if bit 6 is 0. SOD is always reset by the RESET IN signal.

```
Opcode: 0 0 1 1 0 0 0 0

Accumulator Content
Before SIM:

<table>
<thead>
<tr>
<th>SOD</th>
<th>SOE</th>
<th>X</th>
<th>R7.5</th>
<th>MSE</th>
<th>M7.5</th>
<th>M6.5</th>
<th>M5.5</th>
</tr>
</thead>
</table>

Cycles: 1
States: 4
Flags: none
```
**8080A/8085A INSTRUCTION SET INDEX**

**Table 5-1**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Bytes</th>
<th>T States</th>
<th>Machine Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACI DATA</td>
<td>CE data</td>
<td>2 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC REG</td>
<td>1000 1SS</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC M</td>
<td>8E</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADO REG</td>
<td>1000 0SS</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADO M</td>
<td>8E</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADI DATA</td>
<td>C6 data</td>
<td>2 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANA REG</td>
<td>1010 0SS</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANA M</td>
<td>A6</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANI DATA</td>
<td>E6 data</td>
<td>2 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL LABEL</td>
<td>CD addr</td>
<td>3 18 17 S R R W*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC LABEL</td>
<td>DC addr</td>
<td>3 9/16 11/17 S R/S R R W*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CM LABEL</td>
<td>FC addr</td>
<td>3 9/16 11/17 S R/S R R W*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMA</td>
<td>2F</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMX</td>
<td>3F</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP REG</td>
<td>1011 1SS</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP M</td>
<td>8E</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNZ LABEL</td>
<td>D4 addr</td>
<td>3 9/16 11/17 S R/S R R W*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CP LABEL</td>
<td>EC addr</td>
<td>3 9/16 11/17 S R/S R R W*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPI DATA</td>
<td>FE data</td>
<td>2 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPPO LABEL</td>
<td>E4 addr</td>
<td>3 9/16 11/17 S R/S R R W*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CZ LABEL</td>
<td>CC addr</td>
<td>3 9/16 11/17 S R/S R R W*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAA</td>
<td>27</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAD RP</td>
<td>00RP 1001</td>
<td>1 10 10 F B B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC REG</td>
<td>00SSS 101</td>
<td>1 4 5 F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC M</td>
<td>35</td>
<td>1 10 10 F R W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEX RP</td>
<td>00RP 1011</td>
<td>1 6 5 S*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EI</td>
<td>F3</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EI</td>
<td>FB</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HLT</td>
<td>76</td>
<td>1 7 7 F B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IN PORT</td>
<td>08 data</td>
<td>2 10 10 F R I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INR REG</td>
<td>00SSS 100</td>
<td>1 4 5 F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INR M</td>
<td>34</td>
<td>1 10 10 F R W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INX RP</td>
<td>00RP 0011</td>
<td>1 6 5 S*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JC LABEL</td>
<td>DA addr</td>
<td>3 7/10 10 F R/F R F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JM LABEL</td>
<td>FA addr</td>
<td>3 7/10 10 F R/F R F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JNC LABEL</td>
<td>C3 addr</td>
<td>3 10 10 F R R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JNZ LABEL</td>
<td>C2 addr</td>
<td>3 7/10 10 F R/F R F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPL LABEL</td>
<td>F2 addr</td>
<td>3 7/10 10 F R/F R F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPE LABEL</td>
<td>EA addr</td>
<td>3 7/10 10 F R/F R F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JPO LABEL</td>
<td>E2 addr</td>
<td>3 7/10 10 F R/F R F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JZ LABEL</td>
<td>CA addr</td>
<td>3 7/10 10 F R/F R F*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA ADDR</td>
<td>3A addr</td>
<td>3 13 13 F R R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDAX RP</td>
<td>00DX 1010</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LHLD ADDR</td>
<td>2A addr</td>
<td>3 16 16 F R R R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LXI</td>
<td>RP,DATA16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>REG,REG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>M,REG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>REG,M</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVI</td>
<td>REG,DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVI</td>
<td>M,DATA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>00</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORA</td>
<td>REG</td>
<td>1011 0SS</td>
<td>1 4 4 F</td>
<td></td>
</tr>
<tr>
<td>ORA</td>
<td>M</td>
<td>86</td>
<td>1 7 7 F R</td>
<td></td>
</tr>
<tr>
<td>ORI</td>
<td>DATA</td>
<td>F6 data</td>
<td>2 7 7 F R</td>
<td></td>
</tr>
<tr>
<td>OUT PORT</td>
<td>D3 data</td>
<td>2 10 10 F R D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCHL</td>
<td>EB</td>
<td>1 6 5 S*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>RP</td>
<td>11RP 0001</td>
<td>1 10 10 F R R</td>
<td></td>
</tr>
<tr>
<td>PUSH</td>
<td>RP</td>
<td>11RP 1010</td>
<td>1 12 11 S W*</td>
<td></td>
</tr>
<tr>
<td>RAL</td>
<td>17</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAR</td>
<td>1F</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RC</td>
<td>D8</td>
<td>1 1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td>C9</td>
<td>1 10 10 F R R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RIM (8085A only)</td>
<td>20</td>
<td>1 4 - F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLC</td>
<td>07</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RM</td>
<td>F8</td>
<td>1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNC</td>
<td>D0</td>
<td>1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNZ</td>
<td>CD</td>
<td>1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RP</td>
<td>F0</td>
<td>1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPE</td>
<td>EB</td>
<td>1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPO</td>
<td>E0</td>
<td>1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRC</td>
<td>OF</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RST</td>
<td>N</td>
<td>11XX X11</td>
<td>1 12 11 S W*</td>
<td></td>
</tr>
<tr>
<td>RR</td>
<td>C8</td>
<td>1 6/12 5/11 S R/S R*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBB REG</td>
<td>1001 1SS</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBB M*</td>
<td>55</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBI DATA</td>
<td>DE data</td>
<td>2 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIM (8085A only)</td>
<td>30</td>
<td>1 4 - F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPHL</td>
<td>F9</td>
<td>1 6 5 S*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA ADDR</td>
<td>32 addr</td>
<td>3 13 13 F R R W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAX RP</td>
<td>000X 0010</td>
<td>1 7 7 F W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STC</td>
<td>37</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB REG</td>
<td>1001 0SSS</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB M</td>
<td>96</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUI DATA</td>
<td>D5 data</td>
<td>2 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XCHG</td>
<td>EB</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XRA REG</td>
<td>1010 1SSS</td>
<td>1 4 4 F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XRA M</td>
<td>AE</td>
<td>1 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XRI DATA</td>
<td>EE data</td>
<td>2 7 7 F R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XTHL</td>
<td>E3</td>
<td>1 16 18 F R/R W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Machine cycle types:

- F: Four clock period instr fetch
- S: Six clock period instr fetch
- R: Memory read
- I: I/D read
- W: Memory write
- D: I/O write
- B: Bus idle
- X: Variable or optional binary digit

**DDD**: Binary digits identifying a destination register

| B = 000, C = 001, D = 010 Memory = 110 |

**SSS**: Binary digits identifying a source register

| E = 011, H = 100, L = 101 A = 111 |

**RP**: Register Pair

| BC = 00, HL = 10 |
| DE = 01, SP = 11 |

*Five clock period instruction fetch with 8080A.*

*The longer machine cycle sequence applies regardless of condition evaluation with 8080A.*

*An extra READ cycle (R) will occur for this condition with 8080A.*

Mnemonics copyrighted © Intel Corporation 1976.
## 8085A CPU INSTRUCTIONS IN OPERATION CODE SEQUENCE

Table 5-2

<table>
<thead>
<tr>
<th>OP CODE</th>
<th>MNEMONIC</th>
<th>OP CODE</th>
<th>MNEMONIC</th>
<th>OP CODE</th>
<th>MNEMONIC</th>
<th>OP CODE</th>
<th>MNEMONIC</th>
<th>OP CODE</th>
<th>MNEMONIC</th>
<th>OP CODE</th>
<th>MNEMONIC</th>
<th>OP CODE</th>
<th>MNEMONIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NOP</td>
<td>2B</td>
<td>DCX H</td>
<td>56</td>
<td>MOV D,M</td>
<td>81</td>
<td>ADD C</td>
<td>AC</td>
<td>XRA H</td>
<td>D7</td>
<td>RST</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>LXI B,D16</td>
<td>2C</td>
<td>INR L</td>
<td>57</td>
<td>MOV D,A</td>
<td>82</td>
<td>ADD D</td>
<td>AD</td>
<td>XRA L</td>
<td>D8</td>
<td></td>
<td>RC</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>STAX B</td>
<td>2D</td>
<td>DCR L</td>
<td>58</td>
<td>MOV E,B</td>
<td>83</td>
<td>ADD E</td>
<td>AE</td>
<td>XRA M</td>
<td>D9</td>
<td></td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>INX B</td>
<td>2E</td>
<td>MVI L,D8</td>
<td>59</td>
<td>MOV E,C</td>
<td>84</td>
<td>ADD H</td>
<td>AF</td>
<td>XRA A</td>
<td>DA</td>
<td></td>
<td>JC</td>
<td>Adr</td>
</tr>
<tr>
<td>04</td>
<td>INR B</td>
<td>2F</td>
<td>CMA</td>
<td>6A</td>
<td>MOV E,D</td>
<td>85</td>
<td>ADD L</td>
<td>B0</td>
<td>ORA B</td>
<td>DB</td>
<td></td>
<td>IN</td>
<td>D8</td>
</tr>
<tr>
<td>05</td>
<td>DCR B</td>
<td>30</td>
<td>SIM</td>
<td>5B</td>
<td>MOV E,E</td>
<td>86</td>
<td>ADD M</td>
<td>B1</td>
<td>ORA C</td>
<td>DC</td>
<td></td>
<td>CC</td>
<td>Adr</td>
</tr>
<tr>
<td>06</td>
<td>MVI B,D8</td>
<td>31</td>
<td>LXI SP,D16</td>
<td>6C</td>
<td>MOV E,H</td>
<td>87</td>
<td>ADD A</td>
<td>B2</td>
<td>ORA D</td>
<td>DD</td>
<td></td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>RLC</td>
<td>32</td>
<td>STA Addr</td>
<td>5D</td>
<td>MOV E,L</td>
<td>88</td>
<td>ADC B</td>
<td>B3</td>
<td>ORA E</td>
<td>DE</td>
<td></td>
<td>SBI D8</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>–</td>
<td>33</td>
<td>INX SP</td>
<td>5E</td>
<td>MOV E,M</td>
<td>89</td>
<td>ADC C</td>
<td>B4</td>
<td>ORA H</td>
<td>DF</td>
<td></td>
<td>RST</td>
<td>3</td>
</tr>
<tr>
<td>09</td>
<td>DAD B</td>
<td>34</td>
<td>INR M</td>
<td>5F</td>
<td>MOV E,A</td>
<td>90</td>
<td>ADC D</td>
<td>B5</td>
<td>ORA L</td>
<td>E0</td>
<td></td>
<td>RPO</td>
<td></td>
</tr>
<tr>
<td>0A</td>
<td>LDAX B</td>
<td>35</td>
<td>DCR M</td>
<td>60</td>
<td>MOV H,B</td>
<td>8B</td>
<td>ADC E</td>
<td>B6</td>
<td>ORA M</td>
<td>E1</td>
<td></td>
<td>POP</td>
<td>H</td>
</tr>
<tr>
<td>0B</td>
<td>DCX B</td>
<td>36</td>
<td>MVI M,D8</td>
<td>61</td>
<td>MOV H,C</td>
<td>8C</td>
<td>ADC H</td>
<td>B7</td>
<td>ORA A</td>
<td>E2</td>
<td></td>
<td>JPO</td>
<td>Adr</td>
</tr>
<tr>
<td>0C</td>
<td>INR C</td>
<td>37</td>
<td>STC</td>
<td>62</td>
<td>MOV H,D</td>
<td>8D</td>
<td>ADC L</td>
<td>B8</td>
<td>CMP B</td>
<td>E3</td>
<td></td>
<td>XTHL</td>
<td></td>
</tr>
<tr>
<td>0D</td>
<td>DCR C</td>
<td>38</td>
<td>–</td>
<td>63</td>
<td>MOV H,E</td>
<td>8E</td>
<td>ADC M</td>
<td>B9</td>
<td>CMP C</td>
<td>E4</td>
<td></td>
<td>CPO</td>
<td>Adr</td>
</tr>
<tr>
<td>0E</td>
<td>MVI C,D8</td>
<td>39</td>
<td>DAD SP</td>
<td>64</td>
<td>MOV H,H</td>
<td>8F</td>
<td>ADC A</td>
<td>BA</td>
<td>CMP D</td>
<td>E5</td>
<td></td>
<td>PUSH</td>
<td>H</td>
</tr>
<tr>
<td>0F</td>
<td>RRC</td>
<td>3A</td>
<td>LDA Addr</td>
<td>65</td>
<td>MOV H,L</td>
<td>90</td>
<td>SUB B</td>
<td>BB</td>
<td>CMP E</td>
<td>E6</td>
<td></td>
<td>ANI</td>
<td>D8</td>
</tr>
<tr>
<td>10</td>
<td>–</td>
<td>3B</td>
<td>DCX SP</td>
<td>66</td>
<td>MOV H,M</td>
<td>91</td>
<td>SUB C</td>
<td>BC</td>
<td>CMP F</td>
<td>E7</td>
<td></td>
<td>RST</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>LXI D,D16</td>
<td>3C</td>
<td>INR A</td>
<td>67</td>
<td>MOV H,A</td>
<td>92</td>
<td>SUB D</td>
<td>BD</td>
<td>CMP G</td>
<td>EB</td>
<td></td>
<td>RPE</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>STAX D</td>
<td>3D</td>
<td>DCR A</td>
<td>68</td>
<td>MOV L,B</td>
<td>93</td>
<td>SUB E</td>
<td>BE</td>
<td>CMP H</td>
<td>EB</td>
<td></td>
<td>PCHL</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>INX D</td>
<td>3E</td>
<td>MVI A,D8</td>
<td>69</td>
<td>MOV L,C</td>
<td>94</td>
<td>SUB H</td>
<td>BF</td>
<td>CMP I</td>
<td>EA</td>
<td></td>
<td>JPE</td>
<td>Adr</td>
</tr>
<tr>
<td>14</td>
<td>INR D</td>
<td>3F</td>
<td>CMC</td>
<td>6A</td>
<td>MOV L,D</td>
<td>95</td>
<td>SUB L</td>
<td>C0</td>
<td>RINZ</td>
<td>EB</td>
<td></td>
<td>XCHG</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>DCR D</td>
<td>40</td>
<td>MOV B,B</td>
<td>6B</td>
<td>MOV L,E</td>
<td>96</td>
<td>SUB M</td>
<td>C1</td>
<td>POP B</td>
<td>EC</td>
<td></td>
<td>CPE</td>
<td>Adr</td>
</tr>
<tr>
<td>16</td>
<td>MVI D,D8</td>
<td>41</td>
<td>MOV B,C</td>
<td>6C</td>
<td>MOV L,H</td>
<td>97</td>
<td>SUB A</td>
<td>C2</td>
<td>JNZ Addr</td>
<td>ED</td>
<td></td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>RAL</td>
<td>42</td>
<td>MOV B,D</td>
<td>6D</td>
<td>MOV L,L</td>
<td>98</td>
<td>SBB B</td>
<td>C3</td>
<td>JMP Addr</td>
<td>EE</td>
<td></td>
<td>XRI</td>
<td>D8</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>43</td>
<td>MOV B,E</td>
<td>6E</td>
<td>MOV L,M</td>
<td>99</td>
<td>SBB C</td>
<td>C4</td>
<td>CNZ Addr</td>
<td>EF</td>
<td></td>
<td>RST</td>
<td>5</td>
</tr>
<tr>
<td>19</td>
<td>DAD D</td>
<td>44</td>
<td>MOV B,M</td>
<td>6F</td>
<td>MOV L,A</td>
<td>9A</td>
<td>SBB D</td>
<td>C5</td>
<td>PUSH B</td>
<td>F0</td>
<td></td>
<td>RP</td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>LDAX D</td>
<td>45</td>
<td>MOV B,L</td>
<td>70</td>
<td>MOV M,B</td>
<td>9B</td>
<td>SBB E</td>
<td>C6</td>
<td>ADI D</td>
<td>F1</td>
<td></td>
<td>POP</td>
<td>PSW</td>
</tr>
<tr>
<td>1B</td>
<td>DCX D</td>
<td>46</td>
<td>MOV B,M</td>
<td>71</td>
<td>MOV M,C</td>
<td>9C</td>
<td>SBB F</td>
<td>C7</td>
<td>RST D</td>
<td>F2</td>
<td></td>
<td>JP</td>
<td>Adr</td>
</tr>
<tr>
<td>1C</td>
<td>INR E</td>
<td>47</td>
<td>MOV B,A</td>
<td>72</td>
<td>MOV M,D</td>
<td>9D</td>
<td>SBB G</td>
<td>C8</td>
<td>RZ</td>
<td>F3</td>
<td></td>
<td>DI</td>
<td></td>
</tr>
<tr>
<td>1D</td>
<td>DCR E</td>
<td>48</td>
<td>MOV C,B</td>
<td>73</td>
<td>MOV M,E</td>
<td>9E</td>
<td>SBB H</td>
<td>C9</td>
<td>RET Addr</td>
<td>F4</td>
<td></td>
<td>CP</td>
<td>Adr</td>
</tr>
<tr>
<td>1E</td>
<td>MVI E,D8</td>
<td>49</td>
<td>MOV C,C</td>
<td>74</td>
<td>MOV M,H</td>
<td>9F</td>
<td>SBB I</td>
<td>CA</td>
<td>JZ</td>
<td>F5</td>
<td></td>
<td>PUSH</td>
<td>PSW</td>
</tr>
<tr>
<td>1F</td>
<td>RAR</td>
<td>4A</td>
<td>MOV C,CD</td>
<td>75</td>
<td>MOV M,L</td>
<td>90</td>
<td>Aана B</td>
<td>CB</td>
<td>–</td>
<td>–</td>
<td></td>
<td>ORI</td>
<td>D8</td>
</tr>
<tr>
<td>20</td>
<td>RIM</td>
<td>4B</td>
<td>MOV C,E</td>
<td>76</td>
<td>HLT</td>
<td>A1</td>
<td>ANA C</td>
<td>CC</td>
<td>CZ Addr</td>
<td>F7</td>
<td></td>
<td>RST</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>LXI H,D16</td>
<td>4C</td>
<td>MOV C,H</td>
<td>77</td>
<td>MOV M,A</td>
<td>A2</td>
<td>ANA D</td>
<td>CD</td>
<td>CALL Addr</td>
<td>F8</td>
<td></td>
<td>RM</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SHLD Addr</td>
<td>4D</td>
<td>MOV C,L</td>
<td>78</td>
<td>MOV A,B</td>
<td>A3</td>
<td>ANA E</td>
<td>CE</td>
<td>ACI D</td>
<td>F9</td>
<td></td>
<td>SPHL</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>INX H</td>
<td>4E</td>
<td>MOV C,M</td>
<td>79</td>
<td>MOV A,C</td>
<td>A4</td>
<td>ANA H</td>
<td>CF</td>
<td>RST 1</td>
<td>FA</td>
<td></td>
<td>JM</td>
<td>Adr</td>
</tr>
<tr>
<td>24</td>
<td>INR H</td>
<td>4F</td>
<td>MOV C,A</td>
<td>7A</td>
<td>MOV A,D</td>
<td>A5</td>
<td>ANA L</td>
<td>D0</td>
<td>RNC</td>
<td>FB</td>
<td></td>
<td>E1</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>DCR H</td>
<td>50</td>
<td>MOV D,B</td>
<td>7B</td>
<td>MOV A,E</td>
<td>A6</td>
<td>ANA M</td>
<td>D1</td>
<td>POP D</td>
<td>FC</td>
<td></td>
<td>CM</td>
<td>Adr</td>
</tr>
<tr>
<td>26</td>
<td>MVI H,D8</td>
<td>51</td>
<td>MOV D,C</td>
<td>7C</td>
<td>MOV A,H</td>
<td>A7</td>
<td>ANA A</td>
<td>D2</td>
<td>JNC Addr</td>
<td>FD</td>
<td></td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>DAA</td>
<td>52</td>
<td>MOV D,D</td>
<td>7D</td>
<td>MOV A,L</td>
<td>A8</td>
<td>XRA B</td>
<td>D3</td>
<td>OUT D</td>
<td>FE</td>
<td></td>
<td>CPI</td>
<td>D8</td>
</tr>
<tr>
<td>28</td>
<td>–</td>
<td>53</td>
<td>MOV D,E</td>
<td>7E</td>
<td>MOV A,M</td>
<td>A9</td>
<td>XRA C</td>
<td>D4</td>
<td>CNC Addr</td>
<td>FF</td>
<td></td>
<td>RST</td>
<td>7</td>
</tr>
<tr>
<td>29</td>
<td>DAD H</td>
<td>54</td>
<td>MOV D,H</td>
<td>7F</td>
<td>MOV A,A</td>
<td>AA</td>
<td>XRA D</td>
<td>D5</td>
<td>PUSH D</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2A</td>
<td>LHLD Addr</td>
<td>55</td>
<td>MOV D,L</td>
<td>80</td>
<td>ADD B</td>
<td>AB</td>
<td>XRA E</td>
<td>D6</td>
<td>SUI D</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D8 = constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity.

D16 = constant, or logical/arithmetic expression that evaluates to a 16-bit data quantity.

Adr = 16-bit address.

*All mnemonics copyrighted © Intel Corporation 1976.*