# LESSON 8

## 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.

## SYMBOLS AND ABBREVIATIONS:

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

| SYMBOLS | MEANING |
|---|---|
| accumulator | Register A |
| addr | 16-bit address quantity |
| data | 8-bit 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). |
| rm | Bit m of the register r (bits are number 7 through 0 from left to right). |
| | The condition flags: |
| Z | Zero |
| S | Sign |
| P | Parity |
| CY | Carry |
| AC | Auxiliary Carry |
| () | 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.,$(\overline{A})$) |
| n | The restart number 0 through 7 |
| NNN | The binary representation 000 through 111 for restart number 0 through 7 respectively. |

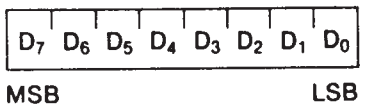
The instruction set encyclopedia is a detailed description of the 8085A instruction set. Each instruction is described in the following manner:

1.The instruction mnemonic and operand fields, are printed on the first line.
2. The name of the instruction is enclosed in parentheses following the mnemonic.
3. The next lines contain a symbolic description of what the instruction does.
4. This is followed by a narrative description of the operation of the instruction.
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.

## INSTRUCTION AND DATA FORMATS

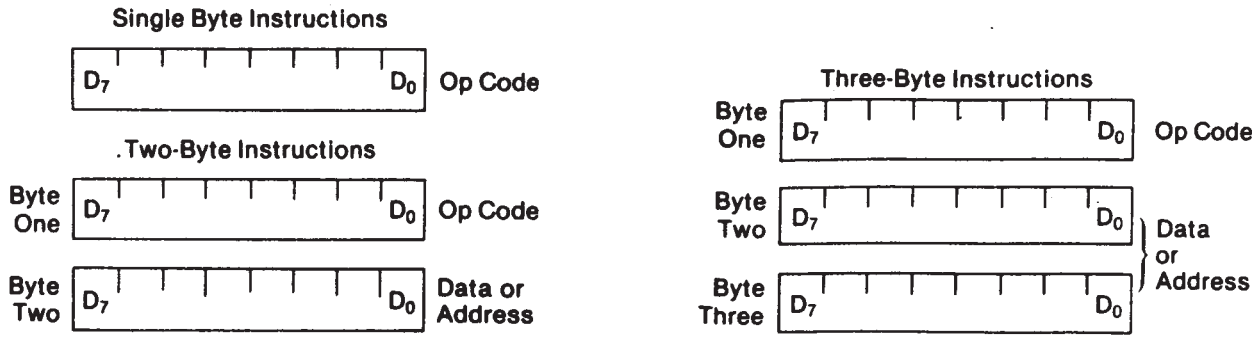
Memory is organized in 8-bit bytes. Each byte has a unique location in physical memory. That location is described by one of a sequence of 16-bit binary addresses. The 8085A can address up to 64K (K = 1024, or $2^{10}$;hence, 64K represents the decimal number 65,536) bytes of memory, which may consist 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:




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 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.




**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. |
| Reg 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.) |
| Reg 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 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; otherwse 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. |
| Aux 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. |

## 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.
3. **Logic Group** - ANDs,ORs,XORs,compares, rotates, or complements data in registers or between memory and a register.
4. **Branch Group** - Initiates conditional or unconditional jumps, calls, returns, and restarts.
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.

### 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.**

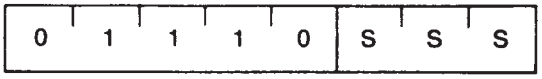
**MOV r1,r2** (Move Register)
(r1) ← (r2)
The content of register r2 is moved to register r1.


Cycles: 1
States: 4
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.

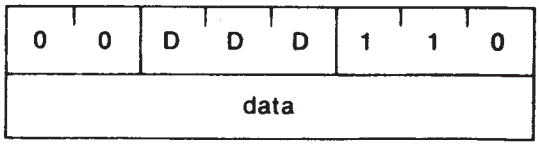

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

**MOV M,r** (Move to memory)
((H)(L)) ← (r)
The content of register r is moved to the memory loction whose address is in registers H and L.



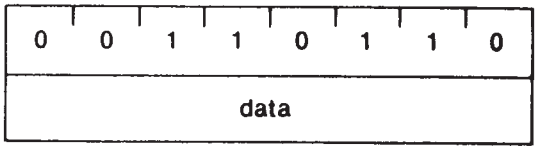

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

**MVI r, data** (Move Immediate)
(r) ← (byte 2)
The content of byte 2 of the instruction is moved to register r.



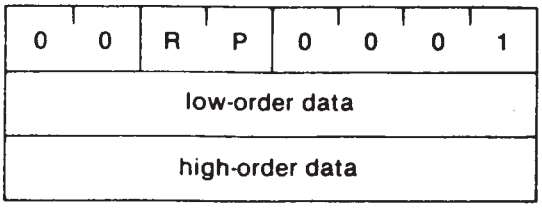

Cycles: 2
States: 7
Addressing: immediate
Flags: none

**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.



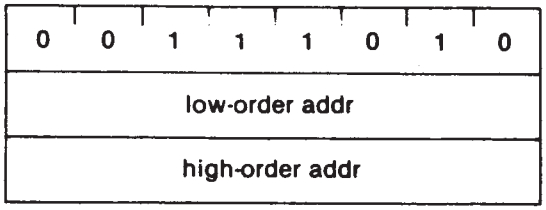

Cycles: 3
States: 10
Addressing: immed./reg. indirect
Flags: none

**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.



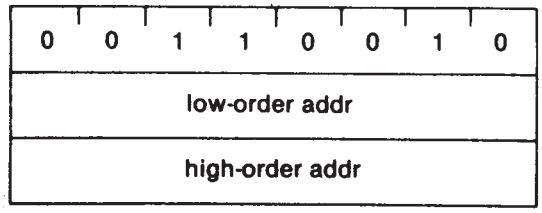

Cycles: 3
States: 10
Addressing: immediate
Flags: none

**LDA addr** (Load Accumulator direct)
(A) ← ((byte 3)(byte2))
The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.



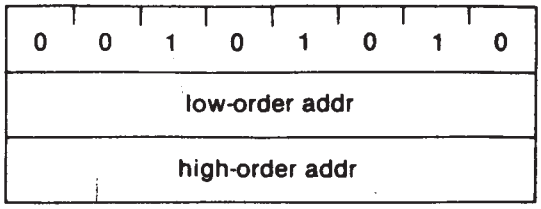

Cycles: 4
States: 13
Addressing: direct
Flags: none

**STA addr** (Store Accumulator direct)
((byte 3)(byte2)) ← (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.



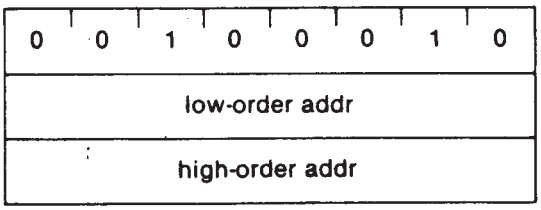

Cycles: 4
States: 13
Addressing: direct
Flags: none

**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

**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.

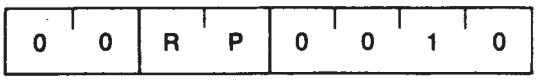



Cycles: 5
States: 16
Addressing: direct
Flags: none

**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.

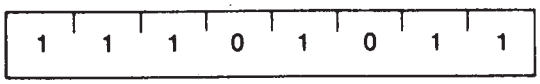

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

**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.




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.



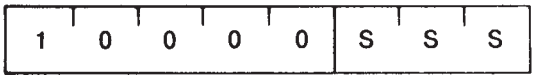

Cycles: 1
States: 4
Addressing: register
Flags: none

**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 Auxillary Carry flags according to the standard rules.**

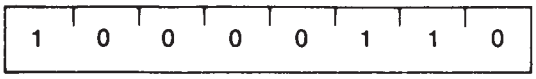
All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

**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.



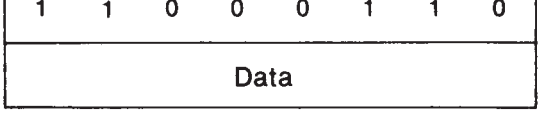

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

**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.




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

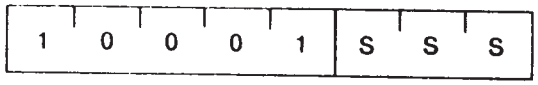
**ADI data** (add immediate)
(A) ← (A) + (byte 2)
The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.




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

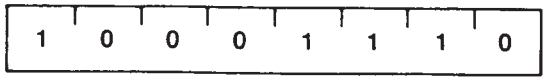
**ADC r** (Add Register with carry)
(A) ← (A) + (r) + (CY)

**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.**



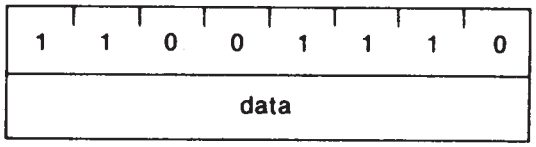

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

**ADC M** (Add memory with carry)
(A) ← (A) + ((H)(L) + (CY)
The content of the memory location whose address is contained in the H and L resisters and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.



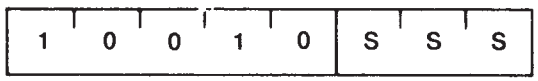

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

**ACI data** (Add immediate with carry)
(A) ← (A) + (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.



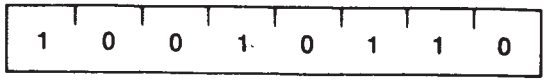

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

**SUB r** (Subtract Register)
(A) ← (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) ← (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.

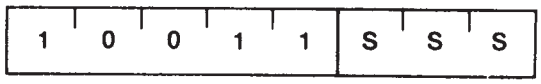



Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

**SUI data** (Subtract immediate)
(A) ← (A) – (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.

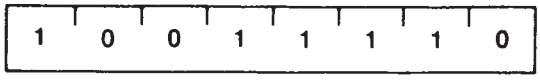

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

**SBB r** (Subtract Register with borrow)
(A) ← (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.




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

**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

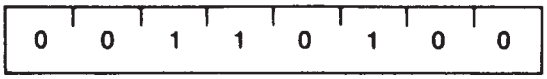
**SBI data** (Subtract immediate with borrow)
(A) ← (A) – (byte 2) – (CY)
The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.


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

**INR r** (Increment Register)
(r) ← (r) + 1
The content of register r is incremented by one. Note condition flags **except CY** are affected.


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

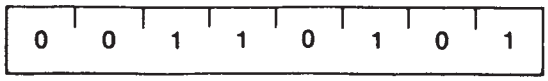



Cycles: 3
States: 10
Addressing: reg. indirect
Flags: Z,S,P,AC

**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
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

**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.**


Cycles: 1
States: 6
Addressing: register
Flags: none

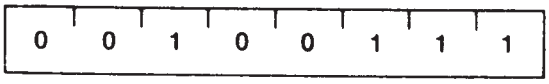
**DCX rp** (Decrement register pair)
(rh)(rl) ← (rh)(rl) – 1
The content of the register pair rp is decremented by one. Note: **No condition flags are affected.**


Cycles: 1
States: 6
Addressing: register
Flags: none

**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.


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 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.

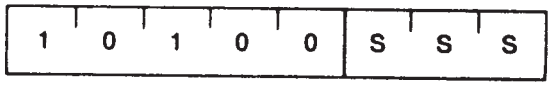



Cycles: 1
States: 4
Flags: Z,S,P,CY,AC

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

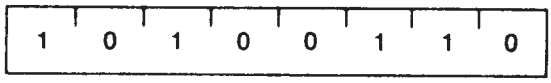
Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

**ANA r** (AND Register)
(A) ← (A)Λ(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.**



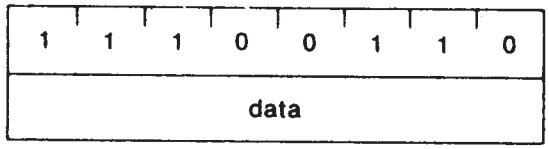

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

**ANA M** (AND) memory
(A) ← (A)Λ((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.**



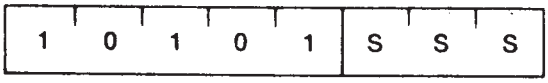

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

**ANI data** (AND immediate)
(A) ← (A)Λ(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.**



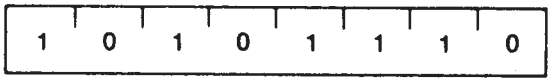

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

**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.**



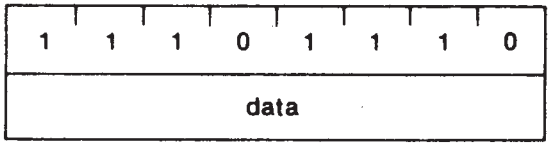

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

**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.**



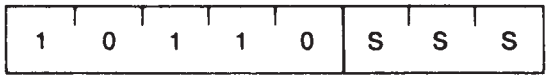

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

**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.**



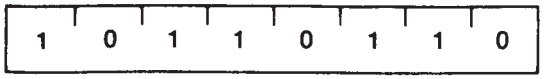

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

**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.**



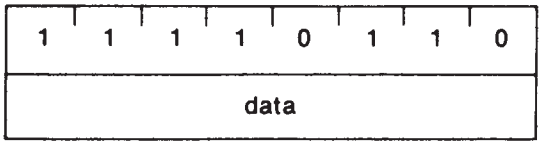

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

**ORA M** (OR memory)
(A) ← (A)V((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) ← (A)V(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

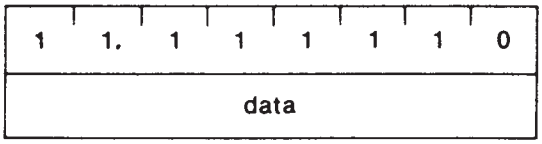
**CMP r** (Compare Register)
(A) – (r)
The content of register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. **The Z flag is set to 1 if (A) = (r).** The CY flag is set to 1 if (A) < (r).


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)).**

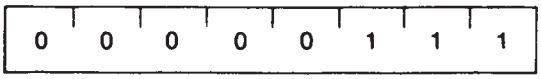

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

**CPI data** (Compare immediate)
(A) – (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) = (byte 2). The CY flag is set to 1 if (A) < (byte 2).**



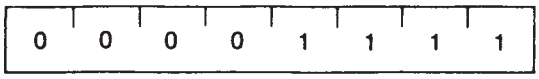

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

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



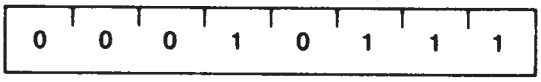

Cycles: 1
States: 4
Flags: CY

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



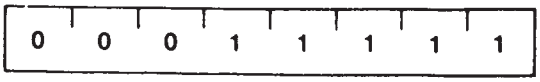

Cycles: 1
States: 4
Flags: CY

**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.**



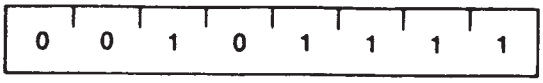

Cycles: 1
States: 4
Flags: CY

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

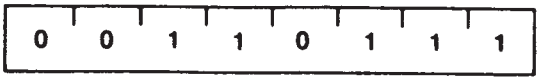



Cycles: 1
States: 4
Flags: none

**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

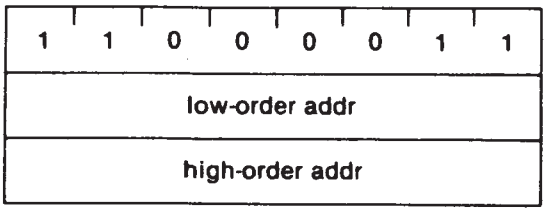
**Branch Group**
This group of instructions alter normal sequential program flow.

**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:

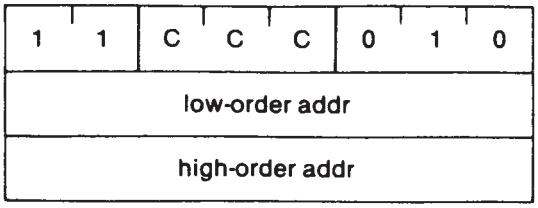
| CONDITION | CCC |
|---|---|
| NZ - not zero (Z = 0) | 000 |
| Z - zero(Z = 1) | 001 |
| NC - no carry (CY = 0) | 010 |
| C - carry (CY = 1) | 011 |
| PO - parity odd (P = 0) | 100 |
| PE - parity even (P = 1) | 101 |
| P - plus (S = 0) | 110 |
| M - minus (S = 1) | 111 |

**JMP addr** (Jump)
(PC) ← (byte 3)(byte 2)
Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.



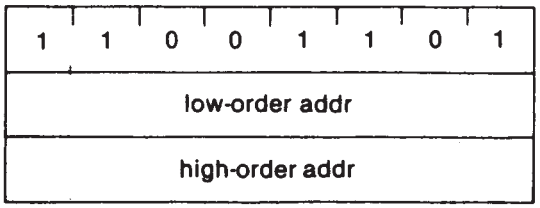

Cycles: 3
States: 10
Addressing: immediate
Flags: none

**J condition 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.



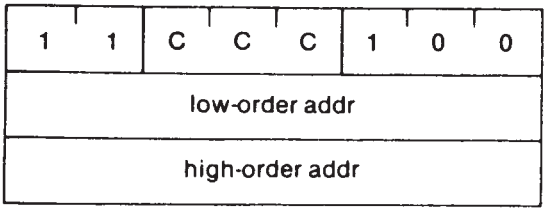

Cycles: 2/3
States: 7/10
Addressing: immediate
Flags: none

**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 by byte 3 and byte 2 of the current instruction.



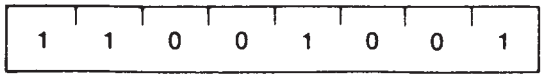

Cycles: 5
States: 18
Addressing: immediate/ reg. indirect
Flags: none

**C condition 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.



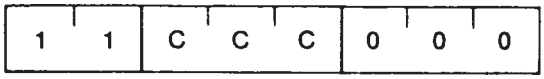

Cycles: 2/5
States: 9/18
Addressing: immediate/ reg. indirect
Flags: none

**RET** (Return)
(PCL)← ((SP));
(PCH)← ((SP) + 1);
(SP)← (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 the register SP is moved to the high-order eight bits of register PC. The content of register SP is incremented by 2.




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

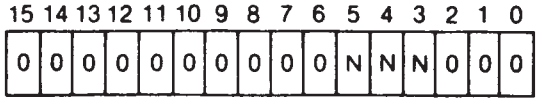
**R condition** (Conditional return)
**If** (CCC),
(PCL)← ((SP))
(PCH)← ((SP) + 1)
(SP)← (SP) = 2
If the specified condition is true, the actions specified in the RET instruction (see above) are performed; otherwise, control continues sequentially.




Cycles: 1/3
States: 6/12
Addressing: reg. indirect
Flags: none

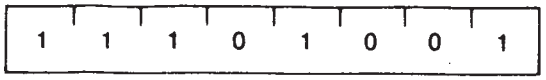
**RST n** (Restart)
((SP) – 1) ← (PCH)
((SP) – 2) ← (PCL)
(SP) ← (SP) – 2
(PC) ← 8 ★ (NNN)
The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are 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
Addressing: reg. indirect
Flags: none




Program Counter After Restart

**PCHL** (Jump H and L indirect-move H and L to PC)
(PCH) ← (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.

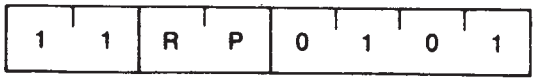



Cycles: 1
States: 6
Addressing: register
Flags: none

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

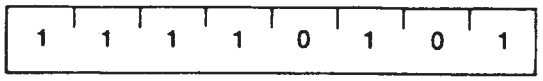
Unless otherwise specified, **condition flags are not affected by any instructions in this group.**

**PUSH rp** (Push)
((SP) – 1) ← (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 or 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.**



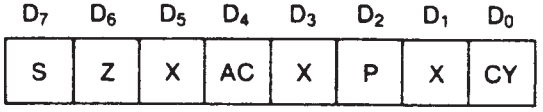

Cycles: 3
States: 12
Addressing: reg. indirect
Flags: none

**PUSH PSW** (Push processor status word)
((SP) – 1) ← (A)
((SP) – 2)$_0$ ← (CY),((SP) – 2)$_1$ ← X
((SP) – 2)$_2$ ← (P), ((SP) – 2)$_3$ ← X
((SP) – 2)$_4$ ← (AC), ((SP) – 2)$_5$ ← X
((SP – 2)$_6$ ← (Z),((SP) – 2)$_7$ ← S
(SP) ← (SP) – 2 X: Undefined.
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.




Cycles: 3
States: 12
Addressing: reg. indirect
Flags: none

FLAG WORD

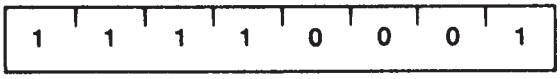



X: undefined

**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 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 rp. The content of register SP is incremented by 2.
**Note: Register pair rp = SP may not be specified.**

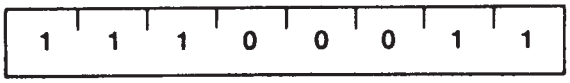

Cycles: 3
States: 10
Addressing: reg.indirect
Flags: none

**POP PSW** (Pop processor status word)
(CY) ← ((SP))$_0$
(P) ← ((SP))$_2$
(AC) ← ((SP))$_4$
(Z) ← ((SP))$_6$
(S) ← ((SP))$_7$
(A) ← ((SP) + 1)
(SP) ← (SP) + 2
The content of the memory location whose address is specified by the content of register SP is used to retore 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.




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

**XTHL** (Exchange stack top with H and L)
(L) ↔ ((SP))
(H) ↔ ((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.

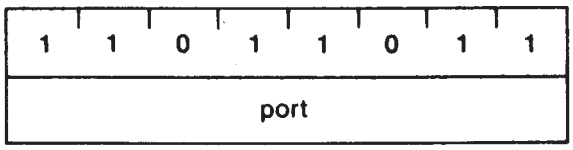



**Cycles:** 5
**States:** 16
**Addressing:** reg. indirect
**Flags:** none

**SPHL** (Move HL to SP)
(SP) ← (H)(L)
The contents of registers H and L (16 bits) are moved to register SP.

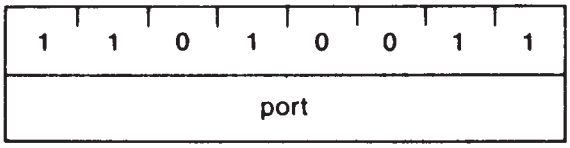

**Cycles:** 1
**States:** 6
**Addressing:** register
**Flags:** none

**IN port** (Input)
(A) ← (data)
The data placed on the eight bit bi-directional data bus by the specified port is moved to register A.



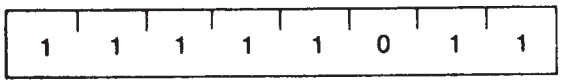

**Cycles:** 3
**States:** 10
**Addressing:** direct
**Flags:** none

**OUT port** (Output)
(data) ← (A)
The content of register A is placed on the eight bit bi-directional data bus for transmission to the specified port.



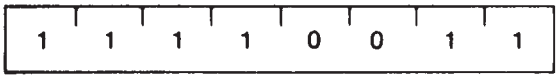

**Cycles:** 3
**States:** 10
**Addressing:** direct
**Flags:** none

**EI** (Enable interrupts)
The interrupt system is enabled **following the execution of the next instruction. Interrupts are not recognized during the EI instruction.**
**NOTE: Placing an EI instruction on the bus in response to $\overline{\text{INTA}}$ during an INA cycle is prohibited.**




Cycles: 1
States: 4
Flags: none

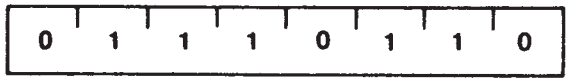
**DI** (Disable interrupts)
The interrupt system is disabled **immediately following the execution of the DI instruction. Interrupts are not recognized during the DI instruction.**




Cycles: 1
States: 4
Flags: none

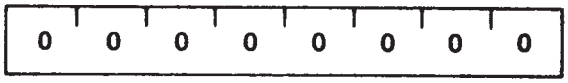
NOTE: Placing a DI instruction on the bus in response to $\overline{\text{INTA}}$ during an INA cycle is prohibited.

**HLT** (Halt)
The processor is stopped. A second ALE is generated during the executive of HLT to strobe out the Halt cycle status information.




**Cycles:** 1+
**States:** 5
**Flags:** none

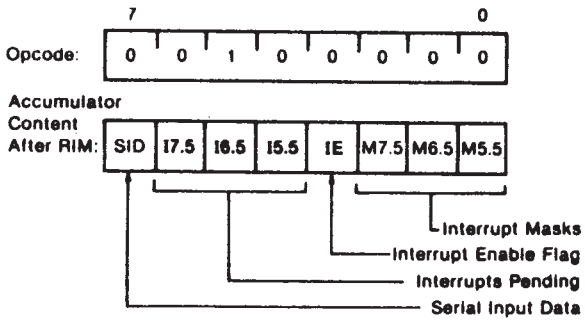
**NOP** (No op)
No operation is performed. The registers and flags are unaffected.




**Cycles:** 1
**States:** 4
**Flags:** none

**RIM** (Read Interrupt Masks)
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.




**Cycles:** 1
**States:** 4
**Flags:** none

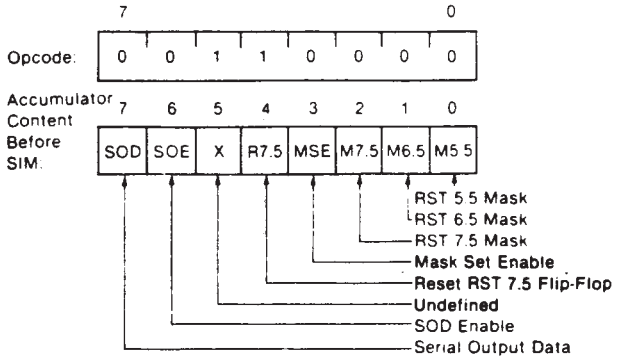
Immediately following a TRAP interrupt, the RIM instruction must be executed as a part of the service routine if you need to retrieve current interrupt status later. Bit 3 of the accumulator is (in this special case only) loaded with the interrupt enable (IE) flag status that existed prior to the TRAP 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.)

**SIM** (Set Interrupt, Masks)
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 $\overline{\text{RESET IN}}$, 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 $\overline{\text{RESET IN}}$ 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 $\overline{\text{RESET IN}}$ signal.




Cycles: 1
States: 4
Flags: none

## Table 8-1 8085A CPU INSTRUCTIONS IN OPERATION CODE SEQUENCE

| OP CODE | MNEMONIC | OP CODE | MNEMONIC | OP CODE | MNEMONIC | OP CODE | MNEMONIC | OP CODE | MNEMONIC | OP CODE | MNEMONIC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | NOP | 28 | DCX H | 56 | MOV D,M | 81 | ADD C | AC | XRA H | D7 | RST 2 |
| 01 | LXI B,D16 | 2C | INR L | 57 | MOV D,A | 82 | ADD D | AD | XRA L | D8 | RC |
| 02 | STAX B | 2D | DCR L | 58 | MOV E,B | 83 | ADD E | AE | XRA M | D9 | - |
| 03 | INX B | 2E | MVI L,D8 | 59 | MOV E,C | 84 | ADD H | AF | XRA A | DA | JC Adr |
| 04 | INR B | 2F | CMA | 5A | MOV E,D | 85 | ADD L | B0 | ORA B | DB | IN D8 |
| 05 | DCR B | 30 | SIM | 5B | MOV E,E | 86 | ADD M | B1 | ORA C | DC | CC Adr |
| 06 | MVI B,D8 | 31 | LXI SP,D16 | 5C | MOV E,H | 87 | ADD A | B2 | ORA D | DD | - |
| 07 | RLC | 32 | STA Adr | 5D | MOV E,L | 88 | ADC B | B3 | ORA E | DE | SBI D8 |
| 08 | - | 33 | INX SP | 5E | MOV E,M | 89 | ADC C | B4 | ORA H | DF | RST 3 |
| 09 | DAD B | 34 | INR M | 5F | MOV E,A | 8A | ADC D | B5 | ORA L | E0 | RPO |
| 0A | LDAX B | 35 | DCR M | 60 | MOV H,B | 8B | ADC E | B6 | ORA M | E1 | POP H |
| 0B | DCX B | 36 | MVI M,D8 | 61 | MOV H,C | 8C | ADC H | B7 | ORA A | E2 | JPO Adr |
| 0C | INR C | 37 | STC | 62 | MOV H,D | 8D | ADC L | B8 | CMP B | E3 | XTHL |
| 0D | DCR C | 38 | - | 63 | MOV H,E | 8E | ADC M | B9 | CMP C | E4 | CPO Adr |
| 0E | MVI C,D8 | 39 | DAD SP | 64 | MOV H,H | 8F | ADC A | BA | CMP D | E5 | PUSH H |
| 0F | RRC | 3A | LDA Adr | 65 | MOV H,L | 90 | SUB B | BB | CMP E | E6 | ANI D8 |
| 10 | - | 3B | DCX SP | 66 | MOV H,M | 91 | SUB C | BC | CMP H | E7 | RST 4 |
| 11 | LXI D,D16 | 3C | INR A | 67 | MOV H,A | 92 | SUB D | BD | CMP L | E8 | RPE |
| 12 | STAX D | 3D | DCR A | 68 | MOV L,B | 93 | SUB E | BE | CMP M | E9 | PCHL |
| 13 | INX D | 3E | MVI A,D8 | 69 | MOV L,C | 94 | SUB H | BF | CMP A | EA | JPE Adr |
| 14 | INR D | 3F | CMC | 6A | MOV L,D | 95 | SUB L | C0 | RNZ | EB | XCHG |
| 15 | DCR D | 40 | MOV B,B | 6B | MOV L,E | 96 | SUB M | C1 | POP B | EC | CPE Adr |
| 16 | MVI D,D8 | 41 | MOV B,C | 6C | MOV L,H | 97 | SUB A | C2 | JNZ Adr | ED | - |
| 17 | RAL | 42 | MOV B,D | 6D | MOV L,L | 98 | SBB B | C3 | JMP Adr | EE | XRI D8 |
| 18 | - | 43 | MOV B,E | 6E | MOV L,M | 99 | SBB C | C4 | CNZ Adr | EF | RST 5 |
| 19 | DAD D | 44 | MOV B,H | 6F | MOV L,A | 9A | SBB D | C5 | PUSH B | F0 | RP |
| 1A | LDAX D | 45 | MOV B,L | 70 | MOV M,B | 9B | SBB E | C6 | ADI D8 | F1 | POP PSW |
| 1B | DCX D | 46 | MOV B,M | 71 | MOV M,C | 9C | SBB H | C7 | RST 0 | F2 | JP Adr |
| 1C | INR E | 47 | MOV B,A | 72 | MOV M,D | 9D | SBB L | C8 | RZ | F3 | DI |
| 1D | DCR E | 48 | MOV C,B | 73 | MOV M,E | 9E | SBB M | C9 | RET Adr | F4 | CP Adr |
| 1E | MVI E,D8 | 49 | MOV C,C | 74 | MOV M,H | 9F | SBB A | CA | JZ | F5 | PUSH PSW |
| 1F | RAR | 4A | MOV C,D | 75 | MOV M,L | A0 | ANA B | CB | - | F6 | ORI D8 |
| 20 | RIM | 4B | MOV C,E | 76 | HLT | A1 | ANA C | CC | CZ Adr | F7 | RST 6 |
| 21 | LXI H,D16 | 4C | MOV C,H | 77 | MOV M,A | A2 | ANA D | CD | CALL Adr | F8 | RM |
| 22 | SHLD Adr | 4D | MOV C,L | 78 | MOV A,B | A3 | ANA E | CE | ACI D8 | F9 | SPHL |
| 23 | INX H | 4E | MOV C,M | 79 | MOV A,C | A4 | ANA H | CF | RST 1 | FA | JM Adr |
| 24 | INR H | 4F | MOV C,A | 7A | MOV A,D | A5 | ANA L | D0 | RNC | FB | EI |
| 25 | DCR H | 50 | MOV D,B | 7B | MOV A,E | A6 | ANA M | D1 | POP D | FC | CM Adr |
| 26 | MVI H,D8 | 51 | MOV D,C | 7C | MOV A,H | A7 | ANA A | D2 | JNC Adr | FD | - |
| 27 | DAA | 52 | MOV D,D | 7D | MOV A,L | A8 | XRA B | D3 | OUT D8 | FE | CPI D8 |
| 28 | | 53 | MOV D,E | 7E | MOV A,M | A9 | XRA C | D4 | CNC Adr | FF | RST 7 |
| 29 | DAD H | 54 | MOV D,H | 7F | MOV A,A | AA | XRA D | D5 | PUSH D | | |
| 2A | LHLD Adr | 55 | MOV D,L | 80 | ADD B | AB | XRA E | D6 | SUI D8 | | |

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.

## Table 8-2 - 8085A INSTRUCTION SET SUMMARY BY FUNCTIONAL GROUPING

| Mnemonic | Description | Instruction Code (1) $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|---|---|
| **MOVE, LOAD, AND STORE** | | | | | | | | | |
| MOVr1 r2 | Move register to register | 0 | 1 | D | D | D | S | S | S |
| MOV M.r | Move register to memory | 0 | 1 | 1 | 1 | 0 | S | S | S |
| MOV r.M | Move memory to register | 0 | 1 | D | D | D | 1 | 1 | 0 |
| MVI r | Move immediate to register | 0 | 0 | D | D | D | 1 | 1 | 0 |
| MVI M | Move immediate to memory | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| LXI B | Load immediate register Pair B & C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| LXI D | Load immediate register Pair D & E | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| LXI H | Load immediate register Pair H & L | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| STAX B | Store A indirect | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| STAX D | Store A indirect | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| LDAX B | Load A indirect | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| LDAX D | Load A indirect | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| STA | Store A direct | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| LDA | Load A direct | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| SHLD | Store H & L direct | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| LHLD | Load H & L direct | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| XCHG | Exchange D & E, H & L registers | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |

# 8085A INSTRUCTION SET SUMMARY (Cont'd)

| Mnemonic | Description | Instruction Code (1) $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|---|---|
| **STACK OPS** | | | | | | | | | |
| PUSH B | Push register Pair B & C on stack | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| PUSH D | Push register Pair D & E on stack | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| PUSH H | Push register Pair H & L on stack | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| PUSH PSW | Push A and Flags on stack | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| POP B | POP register Pair B & C off stack | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| POP D | POP register Pair D & E off stack | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| POP H | POP register Pair H & L off stack | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| POP PSW | POP A and Flags off stack | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| XTHL | Exchange top of stack, H & L | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| SPHL | H & L to stack pointer | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| LXI SP | Load immediate stack pointer | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| INX SP | Increment stack pointer | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| DCX SP | Decrement stack pointer | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| **JUMP** | | | | | | | | | |
| JMP | Jump unconditional | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| JC | Jump on carry | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| JNC | Jump on no carry | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| JZ | Jump on zero | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| JNZ | Jump on no zero | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| JP | Jump on positive | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| JM | Jump on minus | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| JPE | Jump on parity even | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| JPO | Jump on parity odd | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| PCHL | H & L to program counter | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| **CALL** | | | | | | | | | |
| CALL | Call unconditional | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| CC | Call on carry | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| CNC | Call on no carry | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| CZ | Call on zero | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| CNZ | Call on no zero | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| CP | Call on positive | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| CM | Call on minus | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| CPE | Call on parity even | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| CPO | Call on parity odd | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| **RETURN** | | | | | | | | | |
| RET | Return | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| RC | Return on carry | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| RNC | Return on no carry | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| RZ | Return on zero | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| RNZ | Return on no zero | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| RP | Return on positive | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| RM | Return on minus | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| RPE | Return on parity even | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| RPO | Return on parity odd | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| **RESTART** | | | | | | | | | |
| RST | Restart | 1 | 1 | A | A | A | 1 | 1 | 1 |
| **INPUT/OUTPUT** | | | | | | | | | |
| IN | Input | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| OUT | Output | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| **INCREMENT AND DECREMENT** | | | | | | | | | |
| INR r | Increment register | 0 | 0 | D | D | D | 1 | 0 | 0 |
| DCR r | Decrement register | 0 | 0 | D | D | D | 1 | 0 | 1 |
| INR M | Increment memory | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| DCR M | Decrement memory | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| INX B | Increment B & C registers | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| INX D | Increment D & E registers | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| INX H | Increment H & L registers | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

# 8085A INSTRUCTION SET SUMMARY (Cont'd)

| Mnemonic | Description | Instruction Code (1) $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|---|---|
| **INCREMENT AND DECREMENT (cont'd)** | | | | | | | | | |
| DCX B | Decrement B & C | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| DCX D | Decrement D & E | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| DCX H | Decrement H & L | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| **ADD** | | | | | | | | | |
| ADD r | Add register to A | 1 | 0 | 0 | 0 | 0 | S | S | S |
| ADC r | Add register to A with carry | 1 | 0 | 0 | 0 | 1 | S | S | S |
| ADD M | Add memory to A | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| ADC M | Add memory to A with carry | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| ADI | Add immediate to A | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| ACI | Add immediate to A with carry | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| DAD B | Add B & C to H & L | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| DAD D | Add D & E to H & L | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| DAD H | Add H & L to H & L | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| DAD SP | Add stack pointer to H & L | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| **SUBTRACT** | | | | | | | | | |
| SUB r | Subtract register from A | 1 | 0 | 0 | 1 | 0 | S | S | S |
| SBB r | Subtract register from A with borrow | 1 | 0 | 0 | 1 | 1 | S | S | S |
| SUB M | Subtract memory from A | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| SBB M | Subtract memory from A with borrow | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| SUI | Subtract immediate from A | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| SBI | Subtract immediate from A with borrow | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| **LOGICAL** | | | | | | | | | |
| ANA r | And register with A | 1 | 0 | 1 | 0 | 0 | S | S | S |
| XRA r | Exclusive OR register with A | 1 | 0 | 1 | 0 | 1 | S | S | S |
| ORA r | OR register with A | 1 | 0 | 1 | 1 | 0 | S | S | S |
| CMP r | Compare register with A | 1 | 0 | 1 | 1 | 1 | S | S | S |
| ANA M | And memory with A | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| XRA M | Exclusive OR memory with A | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| ORA M | OR memory with A | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| CMP M | Compare memory with A | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| ANI | And immediate with A | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| XRI | Exclusive OR immediate with A | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| ORI | OR immediate with A | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| CPI | Compare immediate with A | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| **ROTATE** | | | | | | | | | |
| RLC | Rotate A left | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| RRC | Rotate A right | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| RAL | Rotate A left through carry | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| RAR | Rotate A right through carry | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| **SPECIALS** | | | | | | | | | |
| CMA | Complement A | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| STC | Set carry | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| CMC | Complement carry | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| DAA | Decimal adjust A | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| **CONTROL** | | | | | | | | | |
| EI | Enable interrupts | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| DI | Disable interrupt | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| NOP | No-operation | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| HLT | Halt | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| **NEW 8085A INSTRUCTIONS** | | | | | | | | | |
| RIM | Read Interrupt Mask | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| SIM | Set Interrupt Mask | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**NOTES:** 1 - DDS or SSS B 000, C 001, D 010, E011, H 100, L 101, Memory 110, A 111

2 - Two possible cycle times. (6/12) indicate instruction cycles dependent on condition flags.