Addressing Modes in 8085

Addressing modes of a microprocessor is the various formats of specifyi operands. Every microprocessor has its own set of instructions. Each of instruction uses one of the addressing modes. 8085 microprocessor has addressing modes. These are :

1. Implied Addressing.
2. Register Addressing.
3. Immediate Addressing.
4. Direct Addressing.
5. Register Indirect Addressing.

1. **Implied Addressing :** The addressing mode of certain 8085 instructi implied by (or inherent in) the instruction's functions. For example, the ST carry flag) instruction deals with the carry flag only and no other regis memory locations.

2. **Register Addressing :** Certain instructions of 8085 specifies operatior operand's source. When the source is register, it is called register address For example, ADD C instruction, in this example source register is C. The cor of C are added to the content of accumulator, as shown below in figure (6

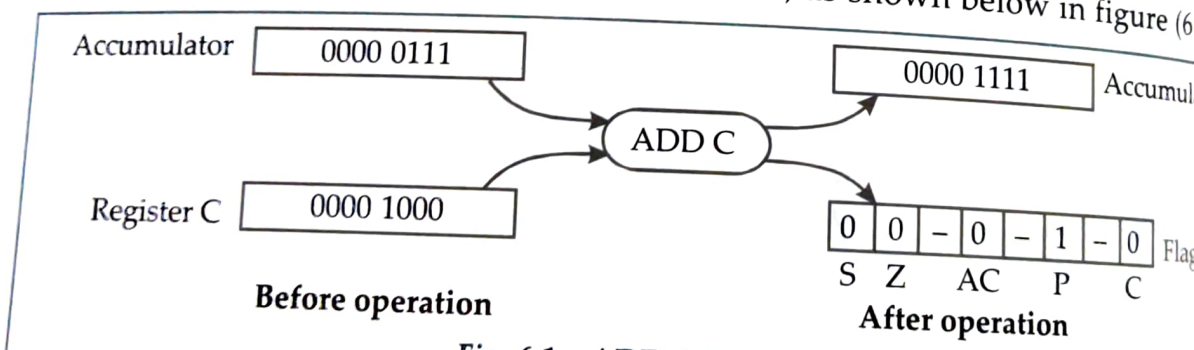


Fig. 6.1 : ADD C instruction

Instructions using register addressing are very efficient in that they only byte of program memory space. They are also executed quickly because the not have to fetch operands from memory.

3. **Immediate Addressing :** In immediate addressing the data appears immedi after op code of instruction in program memory.

For example ADI (add immediate) instruction. This is illustrated in figure

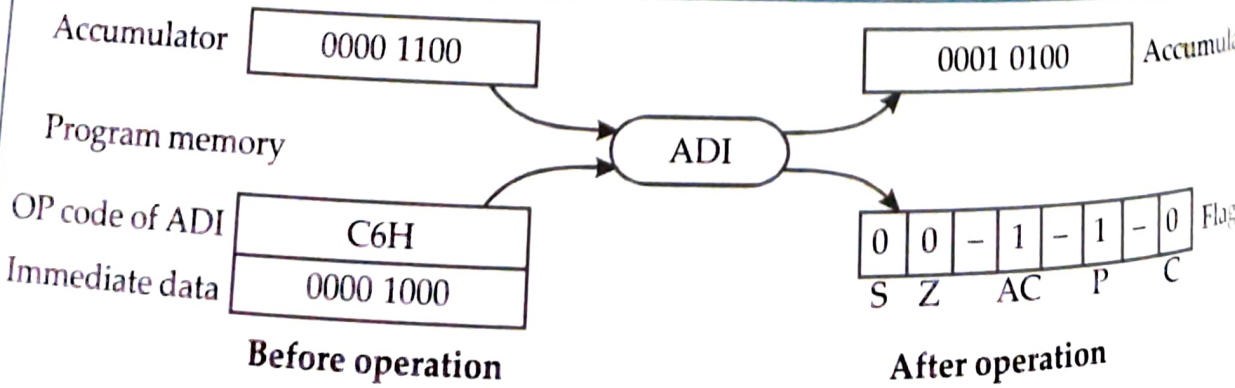


Fig. 6.2 : The ADI instruction

4. **Direct Addressing :** In direct addressing, the address appears after the OP code of instruction in program memory. Instructions using direct addressing are 3 byte instructions. Byte 1 is op code of instruction, byte-2 is low-order address and byte-3 is high order address.

For example, LDA (Load Accumulator direct) instruction.

This is illustrated in figure (6.3).

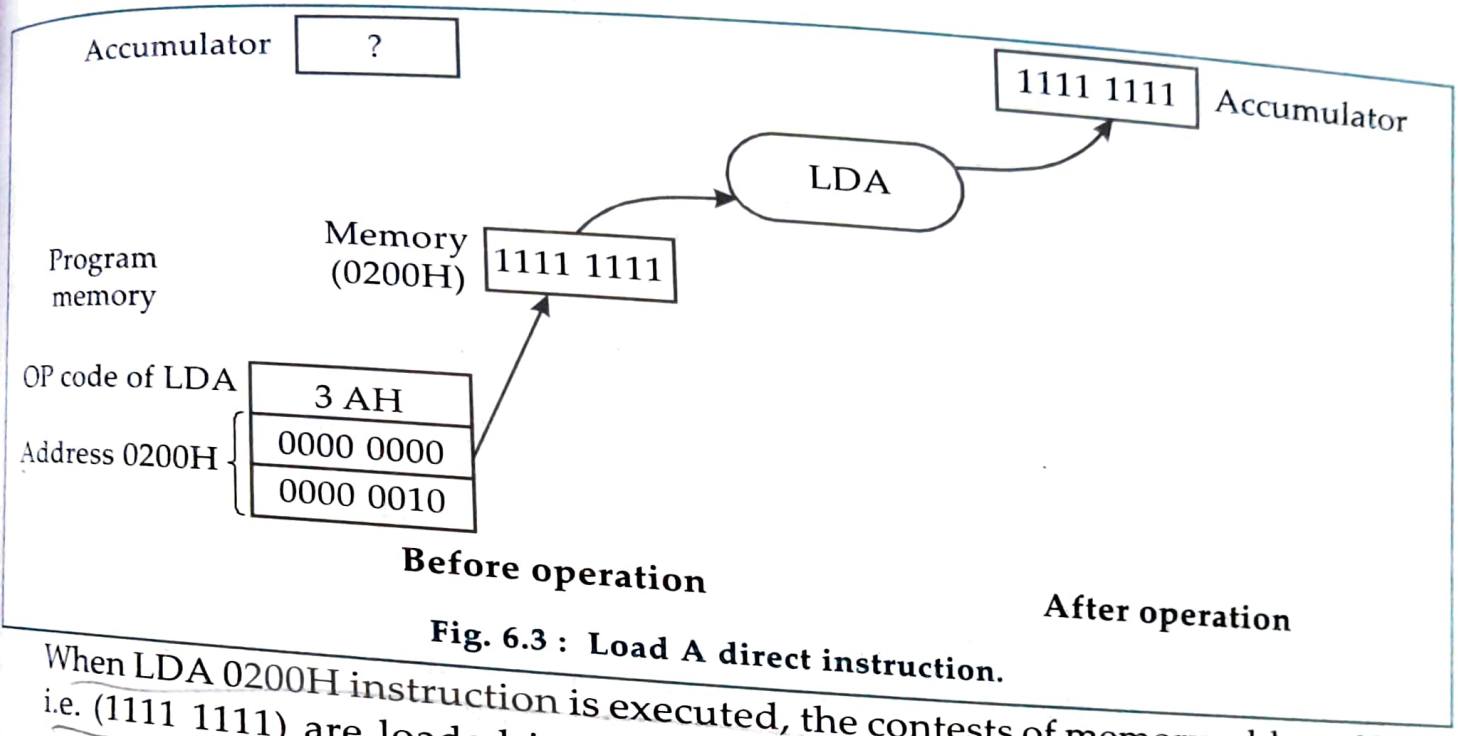


Fig. 6.3 : Load A direct instruction.

When LDA 0200H instruction is executed, the contests of memory address 0200H i.e. (1111 1111) are loaded in accumulator. Direct addressing requires a lot of program memory space.

5. **Register Indirect Addressing :** In register indirect addressing, the contents of register pair points to the address of the operand.

For example, ADD M instruction adds the contents of the accumulator to the contents of memory pointed by the address in HL register pair. This is illustrated in figure (6.4).

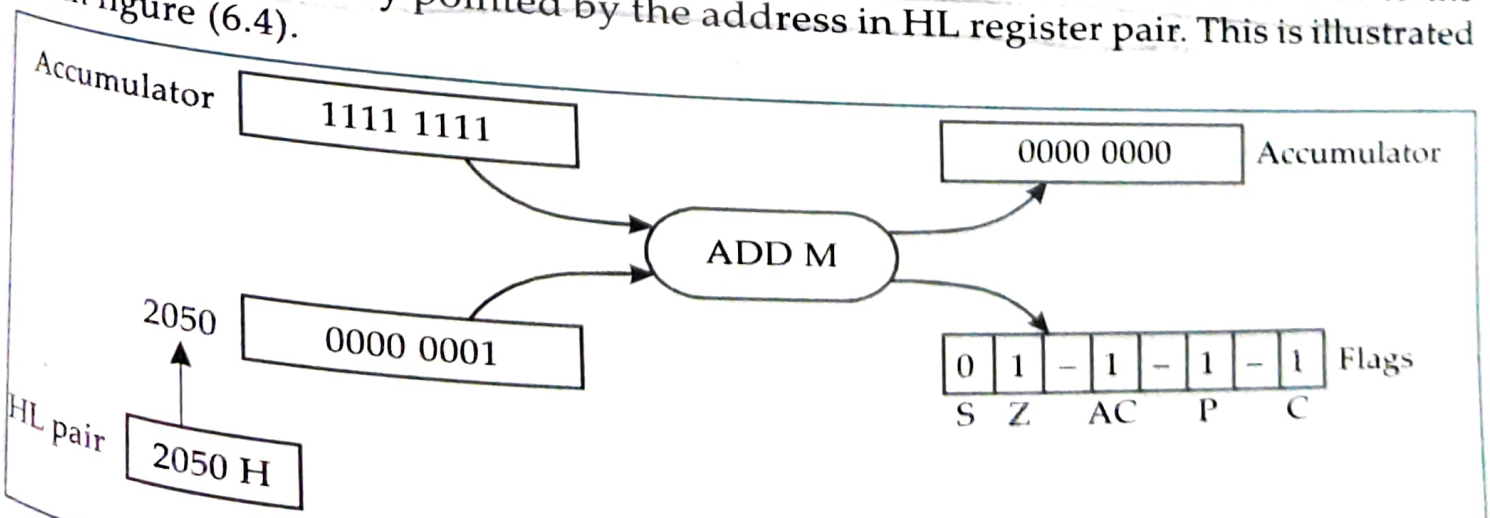


Fig. 6.4 : ADD M instruction

Some of the instructions of 8085 uses combined addressing modes mentioned above. For example CALL instruction combines direct addressing and register indirect addressing.

Programming model of 8085

Programming model of 8085 is a diagram of programmable registers. T programming model of 8085 is shown in figure (6.5). These registers are importa from programmers point of view.

Flags (8)	Accumulator A (8)
B (8)	C (8)
D (8)	E (8)
H (8)	L (8)
(Stock pointer) SP	(16)
(Program counter) PC	(16)

Figure 6.5 Programming model of 8085.

Accumulator is 8 bit register used to perform arithmetic and logical operatio The result of operation is stored in the accumulator. It is also identified as register B, C, D, E, H and L are general purpose registers. They can be combined in pairs B DE, and HL to perform 16–bit operation. The programmer can use these registers store or copy data into registers by using data copy instructions.

Program counter (PC) is a 16–bit register that deals with sequencing the executi of instructions. It contains the address of next instruction to be executed. Stack Poin (SP) is 16 bit register. It points to a memory called stack.

8085 has Five flags. They are Zero (Z), Carry (Cy), Sign (S), Parity (P) a Auxillary Carry (Ac) flag. The microprocessor uses these flags to test data conditio

Instruction Set

An instruction is a binary pattern designed inside a microprocessor to perfo a specific function. The entire group of instructions is called instruction s Instruction set determines what functions the microprocessor can perform. Th instructions can be classified into five functional categories.

1. Data Transfer (Copy) group.
2. Arithmetic group.
3. Logical group.
4. Branching group.
5. Stack, I/O and Machine control group.

The data transfer group of instructions moves data between registers or betw memory locations and registers.

The arithmetic group of instructions performs addition, subtraction, increme or decrements on data in registers or memory.

The logic group of instructions performs ANDs ORs, XORs, Compares, Rot or Complements on data in register or between memory and a register.

The branch group of instructions initiates conditional or unconditional jumps, Calls, Returns and Restarts.

The stack, I/O and machine control group includes instructions for maintaining the stack, reading from input ports, writing to output ports, reading interrupt masks and setting and clearing flags.

Data Transfer Instructions

In data transfer instructions the contents of source are copied into destination without modifying the content of source. The data transfer instructions do not affect the flags. These instructions are mentioned below:

MOV r_1, r_2 (Move register)

$(r_1) \leftarrow (r_2)$, Addressing : register.

The content of register r_2 is moved to register r_1. r_1 and r_2 can be of one of the registers A, B, C, D, E, H, L. This is one byte instruction.

For example: MOV A , B

Contents of B are moved to A (i.e. accumulator).

MOV r, M (Move from memory)

(r) ← (H) (L), Addressing: register indirect. This is one byte instruction. The contents of memory location, whose address is in registers H and L, is moved to register r. r can be any one of the registers A, B, C, D, E, H, L.

For example: MOV B, M

The content of memory pointed by HL pair is moved to B.

MOV M, r (Move to Memory)

((H) (L)) ← (r), Addressing: register indirect

This in one byte instruction.

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

r can be any one of the registers A, B, C, D, E, H, L.

For example: MOV M, C. Here content of register C are moved to memory pointed by HL pair.

MVI r, data (Move Immediate)

(r) ←(byte 2), Addressing: immediate

The contents of byte 2 (data) of instruction is moved to register r. This is two byte instruction.

For example: MVI A, 82 H

The data 82 is moved to accumulator. H in 82H indicates that the number is hexadecimal.

MVI M, data (Move to memory immediate)

((H) (L)) ← (byte 2), Addressing: immediate/reg. indirect.

The content of byte 2 of the instruction is moved to memory location whose addr... byte instruction.

For example: MVI M, 12 H

The data 12 is moved to memory pointed by HL pair.

LXI rp, data 16 (Load register pair immediate)

(rh) ← (byte 3),

(rl) ← (byte 2), Addressing: immediate.

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

The register pair rp can be one of the register pairs BC, DE, HL, or SP. rh is Fi register in pair and rl is second (low order) register in pair.

For example: LXI HL, 4000H.

Here 40H is moved to H and 00H is moved to L.

LDA addr (Load accumulator direct)

(A) ← [(byte 3) (byte 2)], Addressing: direct.

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

For example: LDA C000H. The content of memory location C000H is moved register A.

STA addr (Store Accumulator Direct)

[(byte 3) (byte 2)] ← (A), Addressing: direct.

This is a 3 byte instruction.

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

For example: STA D000 H.

Contents of accumulator are moved to D000.

LHLD addr(Load H and L direct)

(L) ← [(byte 3) (byte 2)]

(H) ← [(byte 3) (byte 2) + 1], Addressing: direct.

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

For example: LHLD 2100 H

The content of 2100 are moved to L and 2101 to H.

SHLD addr (Store H and L direct)

[(byte 3) (byte 2)] ← (L)

[(byte 3) (byte 2) + 1] ← (H), Addressing: direct.

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

For example: SHLD 3520 H.

The content of L are moved to 3520 and that of H to 3521.

LDAX rp (Load accumulator indirect)

(A) ← [(rp)], Addressing: register indirect.

The content of memory location, whose address is in register pair rp, is moved to register A.

rp can be B (i.e. B and C) or D (ie D and E).

This is one byte instruction.

For example : LDAX B

The content of memory location pointed by BC pair is moved to accumulator.

STAX rp (Store accumulator indirect)

[(rp)] ← (A), Addressing: reg. indirect.

The content of register A is moved to the memory location whose address is in register pair rp.

rp is B (ie BC pair) or D (ie DE pair).

This is one byte instruction.

For example : STAX D

The content of A are moved to memory pointed by DE pair.

XCHG (Exchange H and L with D and E)

(H) ⇔ (D)

(L) ⇔ (E) Addressing: register.

The content of register H and L are exchanged with the contents of registers D and E. This is one byte instruction.

Arithmetic Instructions

The 8085 arithmetic instructions performs various arithmetic operations, such as addition, subtraction, increment and decrement. These instructions affects flags and they are discussed below:

ADD r (Add Register)

(A) ← (A) + (r), Addressing : register

The content of register r is added to the contents of the accumulator. The result is placed in accumulator. This is one byte instruction.

For example: ADD B

Content of B are added to contents of accumulator. The result is in accumulator.

ADD M (Add memory)

(A) ← (A) + ((H) (L)), Addressing : reg. indirect.

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

ADI data (Add immediate)

(A) ← (A) + (byte 2), Addressing : immediate.

The content of second byte of the instruction is added to the contents of the accumulator. The result is placed in accumulator. This is a 2 byte instruction.

For example: ADI 05H

The data 05 is added to the content of accumulator. The result is in accumulator.

ADC r (Add register with carry)

(A) ← (A) + (r) + (cy), Addressing : register.

This is one byte instruction. 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 accumulator.

For example: ADC B

Content of B along with carry bit are added to accumulator.

ADC M (Add memory with Carry)

(A) ← (A) + ((H) (L)) + (cy), Addressing reg. indirect.

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

ACI data (Add Immediate with carry)

(A) ← (A) + (byte 2) + (cy), Addressing : immediate.

This is 2 byte instruction. The content of second byte of the instruction and the content of cy flag are added to the contents of accumulator. The result is placed in accumulator.

For example: ACI 12H

The data 12 and carry flag bit is added to the content of accumulator.

SUB r (Subtract Register)

(A) ← (A) – (r), Addressing: register.

This is one byte instruction. The content of register r is subtracted from the content of accumulator. The result is placed in accumulator.

For example: SUB C

Content of C are subtracted from content of accumulator.

SUB M (Subtract memory)

(A) ← (A) – ((H) (L)), Addressing : reg. indirect.

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

SUI data (Subtract immediate)

(A) ← (A) – (byte 2), Addressing :immediate.

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

For example: SUI 05 H

The data 05 is subtracted from the content of accumulator.

SBB r **(Subtract register with borrow)**

(A) ← (A) – (r) – (cy), Addressing : register.

The content of register r and carry bit are subtracted from content of accumulator. The result is placed in accumulator.

This is one byte instruction.

SBB M **(Subtract Memory with borrow)**

(A) ← (A) – [(H) (L)] – (cy), Addressing : reg. indirect.

The content of memory location pointed to by HL register is subtracted from content of accumulator. The carry bit cy is also subtracted from the contents of accumulator. The result is placed in accumulator. This is one byte instruction.

For example: If A = 0000 0100 , cy = 1 and memory location.

Pointed by HL = 00000010 then

```
   0000 00010      Memory location
+         1        Cy
-----------
   0000 0011       New subtrahend

   0000 0100       A
 + 1111 1101       (2's complement of subtrahend)
-----------
 1 0000 0001       difference
 |
 0 ← Cy invert
```

INR r **(Increment register)**

(r) ← (r) + 1 , Addressing: Register.

The content of register r are incremented by one. This is one byte instruction.

All flag except cy flag are affected.

r can be A, B, C, D, E, H, and L.

INR M **(Increment register)**

[(H) (L)] ← [(H) (L)] + 1, Addressing : reg. indirect.

The content of memory location whose address is contained in H and L register is incremented by one. All flag except carry flag are affected.

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

(A) ← (A) – (byte 2) – (cy), Addressing : immediate.

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

DCR r **(Decrement Register)**

(r) ← (r) – 1 , Addressing : Register.

The content of register r is decremented by one. All flags except cy are affected.

For example : DCR B. The contents of B are decremented by one.

DCR M **(Decrement Memory)**

[(H) (L)] ← [(H) (L)] – 1

The content of memory location whose address is contained in the H and registers is decremented by one. All condition flag except cy are affected.

INX rp (Increment register pair)

[(H) (L)] ← [(H) (L)] + 1, Addressing : register.

The content of register pair rp is incremented by one. No condition flag a affected.

For example : 1 INX. The content of HL pair is incremented by one.

DCX rp (Decrement register pair)

[(rh) (r1)] ← [(rh) (rl)] – 1, Addressing : register.

The content of register pair rp is decremented by one.

No condition flags are affected.

For example: DCX H. The content of HL pair is decremented by one.

DAD rp (Add register pair to H and L)

(H) (L) ← (H) (L) + (rh) (rl), Addressing : register.

The content of register pair rp is added to the content of the register pair H a L. The result is placed in register H and L. Only Cy flag is affected.

For example : DAD D. The content of DE are added to HL.

DAA (Decimal Adjust Accumulator)

The Eight bit number in the accumulator is adjusted to form two four – Binary –coded – Decimal digits by the following process.

1. If the value of the least significant 4 bits of the accumulator is grater than 9 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 grater t 9, or if the CY flag is set, 6 is added to the most significant 4 bits of accumulator. All flags are affected.

Logical Instructions

The 8085 logical instructions include AND, OR, XOR, Compare, Rotate Complement, instructions. These instructions are discussed below :

ANA r (AND register)

(A) ← (A) ∧ (r), Addressing : register.

The content of register r is logically AND ed with the content of accumul The result is placed in accumulator. The cy flag is cleared and AC is set. Th one byte instruction.

For example: ANA B

The content of B are logically AND ed with content of A.

If A = FFH = 1111 1111

and B = 02H = 0000 0010

Then ANA B = 0000 0010

ANA M (AND memory)

(A) ← (A) ∧ ((H) (L)) , Addressing : reg. indirect.

The content of 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. This is one byte instruction.

ANI data (AND immediate)

(A) ← (A) ∧ (byte 2), Addressing : immediate.

The content of the second byte of the instruction is logically ANDed with the contents of the accumulator. The result is placed in accumulator. The CY flag is cleared and AC is set.

XRA r (Exclusive OR Register)

(A) ← (A) ∀ (r), Addressing : register.

The content of register r is exclusive OR'ed with the content of accumulator. The result is placed in accumulator. The CY and AC flags are cleared. This is one byte instruction.

For example : If B = 1001 0011 and A = 0001 0101

Then

```
XRA B = 1001 0011
        0001 0101   XOR
---------------
86 H  = 1000 0110
```

XRA M (Exclusive OR memory)

(A) ← (A) ∀ [(H) (L)], Addressing : reg. indirect.

The contents of 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 accumulator.

The CY and AC flags are cleared.

This is one byte instruction.

XRI data (Exclusive OR immediate)

(A) ← (A) ∀ (byte 2), Addressing : immediate.

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

For example: XRI 03H

ORA r (OR register)

(A) ← (A) ∨ [(H) (L)], Addressing : register

The content of the 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.

For example: If B = 1001 0011 (93H)

A = 0001 0101 (15H)

ORA B = 1001 0111 (97H)

ORA M (OR memory)

(A) ← (A) ∨ ((H) (L)), Addressing : reg. indirect.

The content of 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 accumulator. The CY and AC flags are cleared. This is one byte instruction.

ORI data (OR immediate)

(A) ← (A) ∨ (byte 2), Addressing : immediate.

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

This is two byte instruction.

For example: ORI 05 H

CMP r (Compare Register)

(A) – (r) , Addressing : register.

The content of a register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of subtraction. The Z flag is set to 1

if (A) = (r) , The CY flag is set to 1 if (A) < (r).

CMP M (Compare Memory)

(A) – ((H) (L))

The content of 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 is set to 1 if (A) < ((H) (L)).

RLC (Rotate Left)

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

$(CY) \leftarrow (A_7)$

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

Function of RLC is shown in figure (6.6).

For example:

A = 1000 0000 and CY = 0

then after RLC A = 0000 0001 and CY = 1.

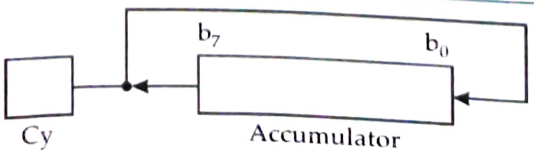


Fig. 6.6 : RLC.

(Rotate Right)

RRC

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

$(CY) \leftarrow (A_0)$.

The content of accumulator is rotated right one position. The higher 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. This is one byte instruction.

Function of RCC is shown in figure (6.7).

For example : If A= 00 00 0001 and CY = 1

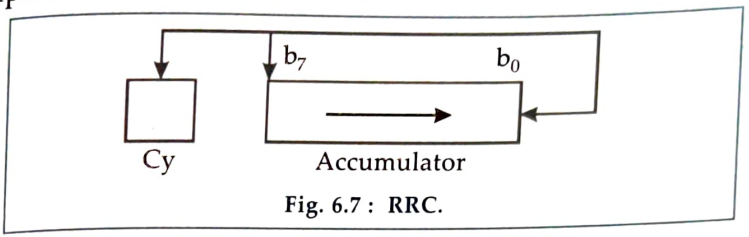


Fig. 6.7 : RRC.

(Rotate left through carry)

RAL

$(A_{n+1}) \leftarrow (A_n)$; $(CY) \leftarrow (A_7)$

$(A_0) \leftarrow (CY)$

The content of 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 of shifted out of the high order bit. Only the cy flag is affected. Function of RAL is shown in figure (6.8).

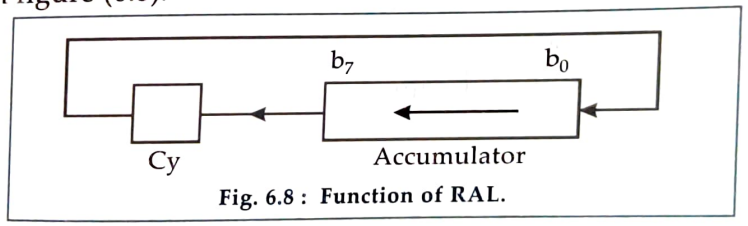


Fig. 6.8 : Function of RAL.

(Rotate Right through carry)

RAR

$(A_n) \leftarrow (A_{n+1})$; $(cy) \leftarrow (A_0)$

$(A_7) \leftarrow (cy)$.

The content of the accumulator is rotated right on 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. Function of RAR is shown in figure (6.9). This is one byte instruction.

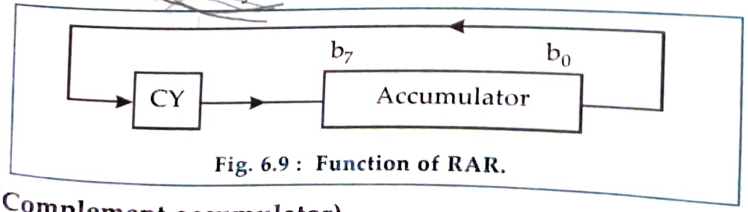


Fig. 6.9 : Function of RAR.

CMA

(Complement accumulator)

$(A) \leftarrow (\overline{A})$

The contents of accumulator are complemented (Zero bits become 1, one bits become 0). No flags are affected.
This is one byte instruction.
For example : If A = 0001 0101 (15H)
then CMA = 1110 1010 (EAH)

CMC (Complement carry)

(cy) ← ($\overline{cy}$)
The carry flag is complemented, No other flags are affected

STC (Set carry)

(cy) ← 1.
Carry flag is set to 1. No other flags are affected.

Branch Instructions

This group of instructions after the normal sequential program flow. There are two types of branch instructions – unconditional and conditional. In unconditional transfer no condition is tested. In conditional transfer the status of one of the flags is tested to determine where the branch is to be executed.

JMP addr (Jump)

(PC) ← (byte 3) (byte 2) , Addressing : immediate.
Control is transferred to the instruction whose address is specified byte 3 and byte 2 of current instruction. This is 3 byte instruction.
For example: JMP 2000 H

Jcondition addr (Conditional jump)

If condition is true,
(PC) ← (byte 3) (byte 2)
Here condition can be NZ, Z, NC, C, PO, PE, P, M
The possible instructions are :

i) JNZ addr .....Jump on Not Zero (Z = 0)
ii) JZ addr .....Jump on Zero (Z = 1)
iii) JNC addr .....Jump on No carry (CY = 0)
iv) JC addr .....Jump on carry (CY = 1)
v) JPO addr .....Jump on odd parity (P = 0)
vi) JPE addr .....Jump on Even parity (P = 1)
vii) JP addr .....Jump on plus (S = 0)
viii)JM addr .....Jump on Minus (S = 1)

In all conditional jumps, the control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction. All the instructions are three byte instructions.

CALL addr (Call)

[(SP) – 1] ← (PCH)

[(SP) – 2] ← (PCL)
(SP) ← (SP) – 2
(PC) ← (byte 3) (byte 2), Addressing : immediate/ reg. indirect.

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 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 current instruction. This is three byte instruction.

Ccondition addr (Conditional call)

If condition is true,
[(SP) – 1] ← (PCH)
[(SP) – 2] ← (PCL)
(SP) ← (SP) – 2
(PC) ← (byte 3) (byte 2)

If the specified condition is true, the action specified in call instructions are performed; otherwise, control continues sequentially. The condition can be any one as specified in conditional jump.

RET (Return)

(PCL) ← [(SP)];
(PCH) ← [(SP) + 1];
(SP) ← (SP) + 2 ; Addressing : reg. indirect.

The content of memory location whose address is specified in register SP is moved to the lower order eight bite of register PC. The content of memory location whose address is one more than the content of memory location whose address is one more than the content of register SP is moved to the high order eight bits of register PC. The content of register SP is incremented by 2. This is one byte instruction.

Rcondition (Conditional return)

If (condition) is ture,
(PCL) ← [(SP)];
(PCH) ← [(SP) + 1];
(SP) ← (SP) + 2 ; Addressing : reg. indirect.

If the specified condition is true, the actions specified in RET are performed; otherwise the control continues sequentially. This is one byte instruction.

RST n (Restart)

[(SP) – 1] ← (PCH)
[(SP) – 2] ← (PCL)
(SP) ← (SP) – 2
(PC) ← 8 *(NNN) Addressing : reg. indirect.

The high order eight bits of next instruction address are moved to the memory location whose address is one less than the content of register SP. The lower order eight bits of the next instruction address are moved to memory location whose address is two less than the contents of register SP. The content of SP is decremented by two. Control is transferred to the instruction whose address is eight times the content of NNN. This is one byte instruction. These instructions are used with interrupts.

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

(PCH) ← (H)
(PCL) ← (L) Addressing : register.

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. This is one byte instruction.

Example of Restart

Consider RST 7 instruction. This is illustrated in figure (6.10).

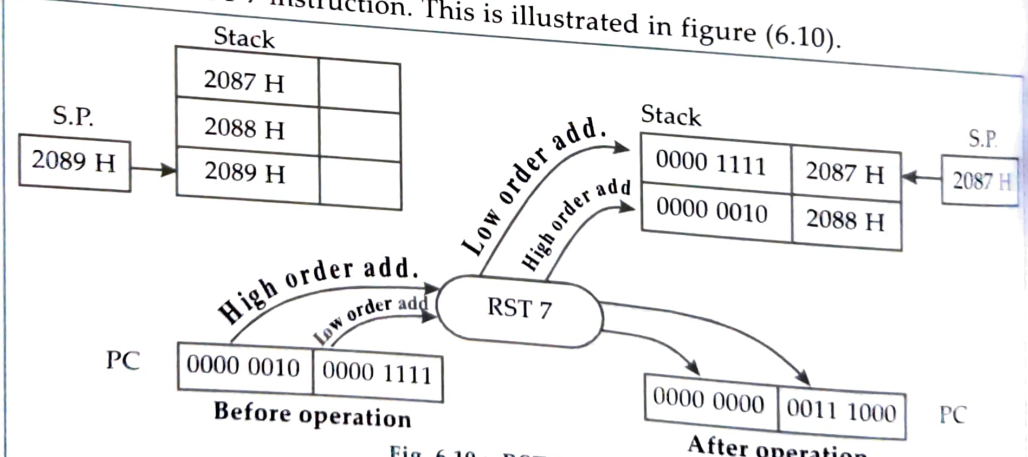


Fig. 6.10 : RST 7 instruction.

RST 7 instruction pushes current program counter into stack. Then the predetermined address ($8 \times 7 = 56_{10} = 38$ H) is transferred to program counter. Restart instructions are called as software interrupts.

Stack, I/O and Machine Control Instructions

This group of instructions performs inputs and outputs, manipulates stack, and alters internal control flags, These instructions are discussed below:

PUSH rp **(Push)**

[(SP) – 1] ← (rh)
[(SP) – 2] ← (rl)
(SP) ← (SP) – 2 Addressing : reg. indirect.

The content of the high order register of register pair rp is moved to 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 SP is decremented by two. This is one byte instruction.

For example : PUSH B

PUSH PSW **(Push processor status word)**

$[(SP) - 1] \leftarrow (A)$

$[(SP) - 2]_0 \leftarrow (cy)$, $[(SP) - 2]_1 \leftarrow x$.

$[(SP) - 2]_2 \leftarrow (P)$, $[(SP) - 2]_3 \leftarrow x$

$[(SP) - 2]_4 \leftarrow (AC)$, $[(SP) - 2]_5 \leftarrow x$

$[(SP) - 2]_6 \leftarrow (z)$, $[(SP) - 2)_7 \leftarrow (S)$

$(SP) \leftarrow (SP) - 2$ x : undefined

Addressing : reg. indirect

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

POP rp **(POP)**

$(rl) \leftarrow [(SP)]$

$(rh) \leftarrow [(SP) + 1]$

$(SP) \leftarrow (SP) + 2$ Addressing : reg. indirect.

The content of memory location, whose address is specified by the content of register of SP, is moved to the low-order register pair rp. The content of 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 SP is incremented by 2, rp = SP may not be specified. This is one byte instruction.

For example: POP B

POP PSW **(POP Processor status word)**

$(CY) \leftarrow [(SP)]_0$

$(P) \leftarrow [(SP)]_2$ $(AC) \leftarrow [(SP)]_4$

$(Z) \leftarrow [(SP)]_6$ $(S) \leftarrow [(SP)]_7$

$(A) \leftarrow [(SP) + 1]$ $(SP) \leftarrow (SP) + 2$

Addressing : reg. indirect.

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

XTHL **(Exchange stack top with H and L)**

$(L) \leftrightarrow [(SP)]$

$(H) \leftrightarrow [(SP) + 1]$

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

SPHL (Move HL to SP)

(SP) ← (H) (L), Addressing : register.

The content of register H and L (16 bits) are moved to register SP. This is one byte instruction. This instruction can be used for initializing the stack pointer.

IN port (Input)

(A) ← (data), Addressing : direct.

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

This is two byte instruction.

For example: IN 10 H, where 10 H is port address.

OUT port (Output)

(data) ← (A), Addressing : direct.

This is two byte instruction.

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

For example : OUT 11 H

EI (Enable Interrupt)

The interrupt system is enabled following the execution of the next instruction. Interrupts are not recognized during the EI instruction. This is one byte instruction.

DI (Disable Interrupts)

The interrupt system is disabled immediately following the execution of DI instruction. Interrupts are not recognised during the DI instruction. This is one byte instruction.

HLT (Halt)

The processor is stopped. The registers and flags are unaffected. This is one byte instruction. This is used to stop MPU. It is waiting for a peripheral device to finish its task and interrupt the processor.

NOP (NO OP)

No operation is performed. The register and flags are unaffected. This is one byte instruction. This instruction is useful as a time delay in a timing loop.

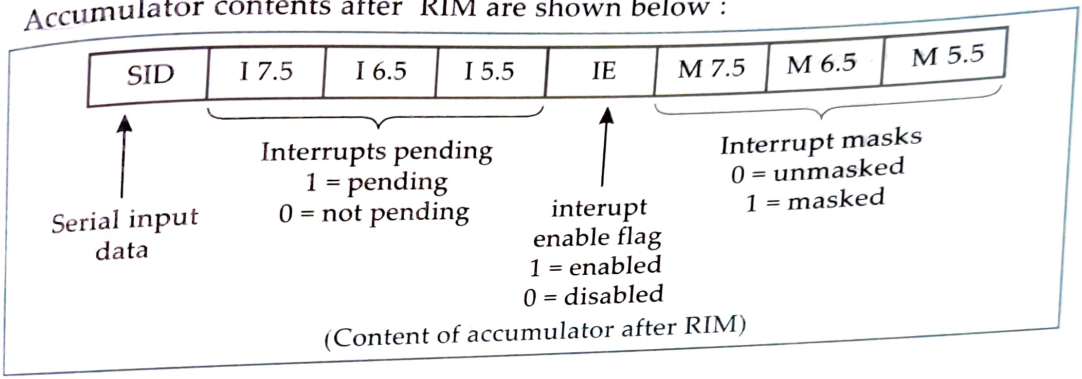
RIM (Read Interrupt Masks)

The RIM instruction loads data into the accumulator relating to interrupts and the serial input. This data contains the following information:

1 Current interrupt mask status for the RST5.5, 6.5 and 7.5 hardware interrupts (1 = Mask disabled)

2. Current interrupt enable flag status (1 = interrupts enabled) except immediately following a TRAP interrupt.
3. Hardware interrupts pending (i.e. signal received but not yet serviced), on the RST 5.5, 6.5 and 7.5 Lines.
4. Serial input data.

Immediately following a TRAP interrupt, the RIM instruction must be executed as a part of service routine if you need to retrieve current interrupt status later. Accumulator contents after RIM are shown below :



(Content of accumulator after RIM)

SIM (Set interrupt masks)

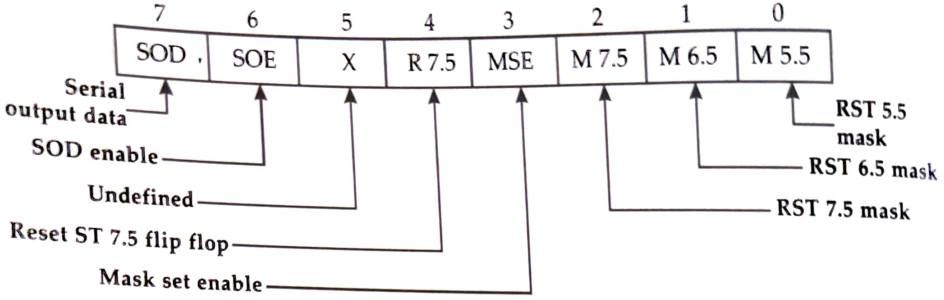
The execution of SIM instruction uses the contents of the accumulator (which must be previously loaded) to perform the following functions:

1. Program the interrupt mask for the RST 5–5, 6.5 and 7.5 hardware interrupts.
2. Reset the edge triggered RST 7.5 input latch.
3. 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 SIM instruction is executed, the interrupt mask register will not change. If bit 4 of accumulator is 1 when 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. The latch remains high until cleared by $\overline{\text{RESETIN}}$, 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 mast (by a SIM instruction). The $\overline{\text{RESETIN}}$, signal always sets all three. RST mask bits.

If accumulator bit 6 is at the 1 level when SIM instruction is executed, the state of accumulator bit 7 is loaded into the SOD latch and thus becomes available for interfaces to external device. The SOD latch is unaffected by the SIM instruction if bit 6 is 0, SOD is always reset by the $\overline{\text{RESETIN}}$ signal.

Accumulator content before SIM



Hardware interrupts of 8085 can be controlled by EI and DI. When EI is executed it will set an interrupt enable flip flop within CPU. When it is set, 8085 will recognize and respond to any of five possible hardware interrupts.

DI instruction will cause interrupt enable flip flop to reset to 0. New 8085 will ignore all interrupts except TRAP interrupt. The TRAP interrupt cannot be disabled by any 8085 program instruction.

EI instruction is frequently used as a part of start up sequence. When power is first turned on, the MPU begins trying to execute instruction from some unknown address. A $\overline{\text{RESET}}$ input is usually applied, forcing Program counter to 0000H. This also causes interrupt enable flag to be reset to 0. The first instruction after RESET is EI.

A **nonmaskable** interrupt is one that the system cannot disable. A **maskable** interrupt is one that the system can disable. TRAP interrupt is nonmaskable while other interrupts are maskable.

The EI and DI instructions enable and disable all the maskable interrupts as a group. Interrupts RST 7.5, RST 6.5, RST 5.5 can be individually enabled or disabled. The actual mechanism for selectively enabling and disabling interrupts is called on interrupt mask. If an interrupt is masked it can not be recognised by the processor.

Consider a simplified interrupt example, shown in figure (6.11).

1. RST 5.5 input (Pin 9) is activated with high signal.
2. To simplify, following things are assumed –
   (a) TRAP is not activated.
   (b) RST 7.5 and RST 6.5 are not activated or masked.
3. Bit b_0 of interrupt status register is checked. It is reset to 0, which satisfies first condition for RST 5.5 to be recognised by MPU.
4. The interrupt enabled flag (b_3) of interrupt register is checked. If is set then which means all interrupts are enabled.
5. RST 5-5 is recognised by MPU. The processor finishes its current instruction and stores the program counter contents in the stack.
6. 8085 issues an interrupt acknowledge ($\overline{\text{INTA}}$) signal to peripherals.
7. The MPU jumps to memory location 002CH.

Memory address	Label	Mnemonic	OP code (hex)	Comments

This is explained as follows :

1. Label is name of line. It is used in jump instruction.
2. Mnemonic consist of instruction and operands.
   For example: MOV A, M.
3. OP code (hex) is a corresponding hexadecimal code for the instruction.
4. Comments are useful for understanding purpose of instructions.
5. Memory address is location where OP code is stored.

Label and comment fields are optional.

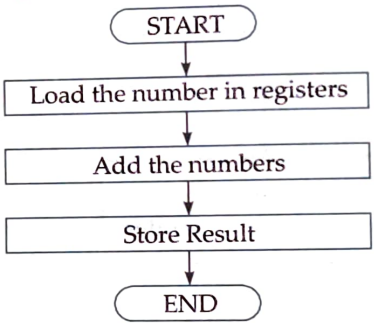
Now lets consider some programming examples.

Problem 6.1: Addition

Write ALP to add two hexadecimal numbers 32H and 48H. Store sum to mem location C000 H.

Solution :

Flowchart for adding two numbers.



Assembly Language Program :

Memory Address	Label	Mnemonic	OP code (Hex)	Comments
5000		MVI A, 32 H	3E	; move 32 in A
01			32	
02				
03		MVI B, 48 H	06	; move 48 in
			48	
04		ADD B	80	; Add content to content o
05		STA C000	32	; store some a
06			00	C000
07			C0	
08		HLT	76	; stop MPU.

Problem 6.2: Ones complement.

Write a ALP to complement the contents of memory location C000H. Place the result in D000H

Solution :

If (C000) = 6A , result = (D000) = 95

Assembly Language Program :

Memory Address	Label	Mnemonic	OP code (Hex)	Comments
5000		LDA C000H	3A	; move contents of
01			00	C000 to A
02			C0	
03		CMA	2F	; Complement A
04		STA D000	32	
05			00	; Store result of D000H
06			D0	
07		HLT	76	; stop MPU.

Problem 6.3: Find larger of two numbers.

Write ALP to find larger of the content of memory locations C000 and C001. Place the result in C002.

Solution :

Assembly Language program :

Memory Address	Label	Mnemonic	OP code (Hex)	Comments
5000		LXI H,C000	21	; initialize H–L
01			00	Pointer
02			C0	
03		MOV A,M	7E	; Get first operand
04		INX H	23	; increment HL pair
05		CMP M	BE	; is second operand larger ?
06		JNC DONE	D2	; jump if not
07			0A	
08			50	
09		MOV A,M	7E	; If yes get second operand in A
0A				
0B	DONE :	INX H	23	; increment HL pair
0C		MOV M, A	77	; store result
		HLT	76	; stop MPU

Problem 6.4: Subtraction

Write ALP to subtract the content of memory location C000 and C001 and place the result in C002.

Solution :

Assembly Language Program :

Memory Address	Label	Mnemonic	OP code (Hex)	Comments
5000		LXI H,C000	21	; initialize HL pointer
01			00	
02			C0	
03		MOV A,M	7E	; Get first operand
04		INX H	23	; increment HL pair
05		SUB M	96	; subtract second operand
06		INX H	23	; increment HL pair
07		MOV M, A	77	; Store result
08		HLT	76	; Stop MPU

Problem 6.5: Maximum in memory block

Find l