TOPIC 1 :Computer Architecture: Instruction Codes

While a **Program**, as we all know, is, A set of instructions that specify the operations, operands, and the sequence by which processing has to occur. An **instruction code** is a group of bits that tells the computer to perform a specific operation part.

Instruction Code: Operation Code

The operation code of an instruction is a group of bits that define operations such as add, subtract, multiply, shift and compliment. The number of bits required for the operation code depends upon the total number of operations available on the computer. The operation code must consist of at least **n bits** for a given **2^n** operations. The operation part of an instruction code specifies the operation to be performed.

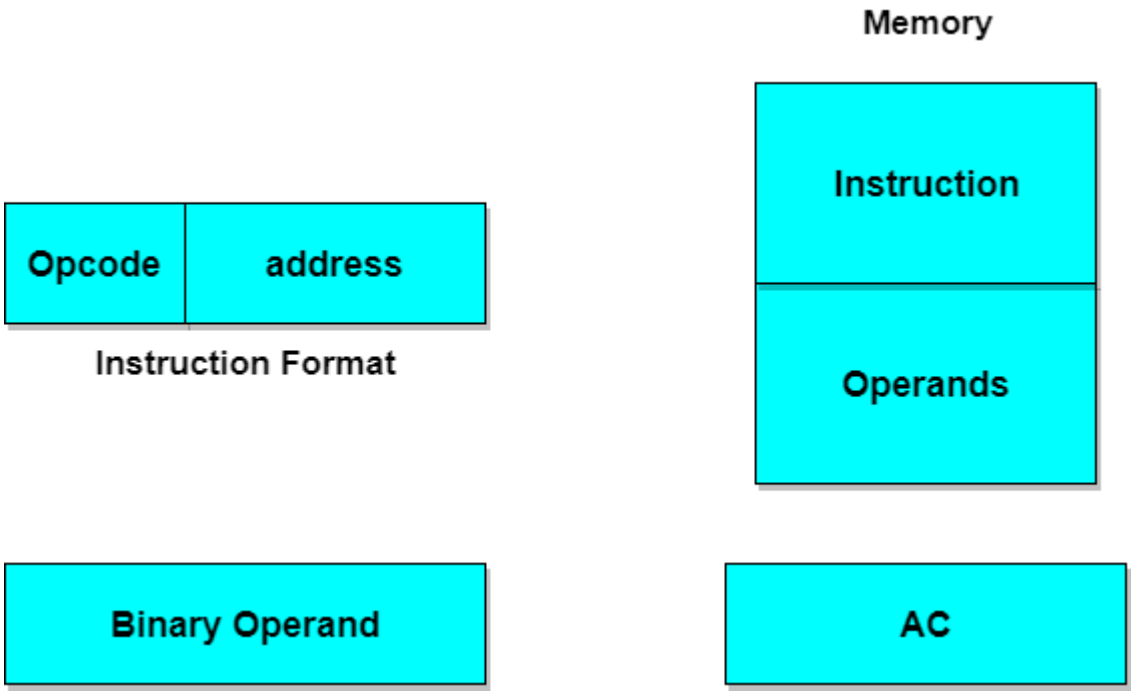
Instruction Code: Register Part

The operation must be performed on the data stored in registers. An instruction code therefore specifies not only operations to be performed but also the registers where the operands(data) will be found as well as the registers where the result has to be stored.

Stored Program Organisation

The simplest way to organize a computer is to have **Processor Register** and instruction code with two parts. The first part specifies the operation to be performed and second specifies an address. The memory address tells where the operand in memory will be found.

Instructions are stored in one section of memory and data in another.



Computers with a single processor register is known as **Accumulator (AC)**. The operation is performed with the memory operand and the content of AC.

Common Bus System

The basic computer has 8 registers, a memory unit and a control unit. Paths must be provided to transfer data from one register to another. An efficient method for transferring data in a system is to use a **Common Bus System**. The output of registers and memory are connected to the common bus.

Load(LD)

The lines from the common bus are connected to the inputs of each register and data inputs of memory. The particular register whose **LD** input is enabled receives the data from the bus during the next *clock pulse transition*.

Before studying about instruction formats lets first study about the operand address parts.

When the 2nd part of an instruction code specifies the operand, the instruction is said to have **immediate operand**. And when the 2nd part of the instruction code specifies the address of an operand, the instruction is said to have a **direct address**. And in **indirect address**, the 2nd part of instruction code, specifies the address of a memory word in which the address of the operand is found.

Computer Instructions

The basic computer has 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

1. **Memory Reference –** These instructions refer to memory address as an operand. The other operand is always accumulator. Specifies 12-bit address, 3-bit opcode (other than 111) and 1-bit addressing mode for direct and indirect addressing.

Example –
IR register contains = 0001XXXXXXXXXXX, i.e. ADD after fetching and decoding of instruction we find out that it is a memory reference instruction for ADD operation.

```
Hence, DR ← M[AR]
AC ← AC + DR, SC ← 0
```

2. **Register Reference –** These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.

Example –
IR register contains = 0111001000000000, i.e. CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.

```
Hence, AC ← ~AC
```

3. **Input/Output –** These instructions are for communication between computer and outside environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.

Example –
IR register contains = 1111100000000000, i.e. INP after fetch and decode cycle we find out that it is an input/output instruction for inputing character. Hence, INPUT character from peripheral device.
The set of instructions incorporated in16 bit IR register are:

1. Arithmetic, logical and shift instructions (and, add, complement, circulate left, right, etc)
2. To move information to and from memory (store the accumulator, load the accumulator)
3. Program control instructions with status conditions (branch, skip)
4. Input output instructions (input character, output character)

Symbol	Hexadecimal Code		Description
AND	0xxx	8xxx	And memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store AC content in memory
BUN	4xxx	Cxxx	Branch Unconditionally
BSA	5xxx	Dxxx	Add memory word to AC
ISZ	6xxx	Exxx	Increment and skip if 0
CLA	7800		Clear AC
CLE	7400		Clear E(overflow bit)
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC > 0
SNA	7008		Skip next instruction if AC < 0
SZA	7004		Skip next instruction if AC = 0
SE	7002		Skip next instruction if E = 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
IEN	F080		Interrupt On

TOPIC 2:Adressing Modes and Instruction Cycle

The operation field of an instruction specifies the operation to be performed. This operation will be executed on some data which is stored in computer registers or the

main memory. The way any operand is selected during the program execution is dependent on the addressing mode of the instruction. The purpose of using addressing modes is as follows:

1. To give the programming versatility to the user.
2. To reduce the number of bits in addressing field of instruction.

Types of Addressing Modes

Below we have discussed different types of addressing modes one by one:

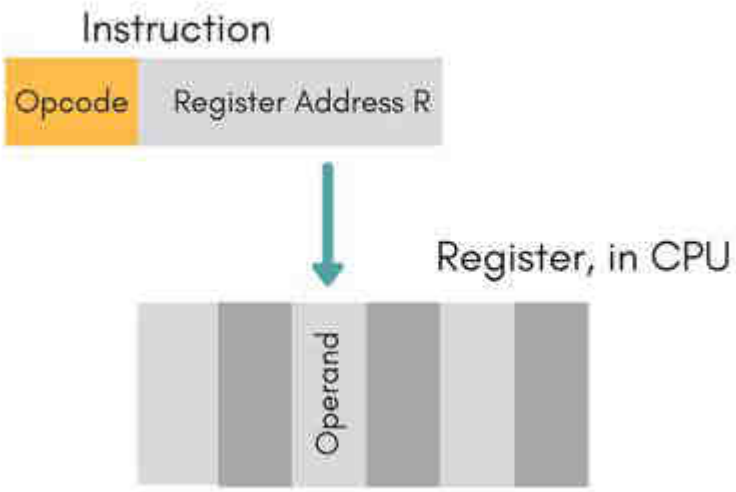
Immediate Mode

In this mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field.

For example: `ADD 7`, which says Add 7 to contents of accumulator. 7 is the operand here.

Register Mode

In this mode the operand is stored in the register and this register is present in CPU. The instruction has the address of the Register where the operand is stored.



Advantages

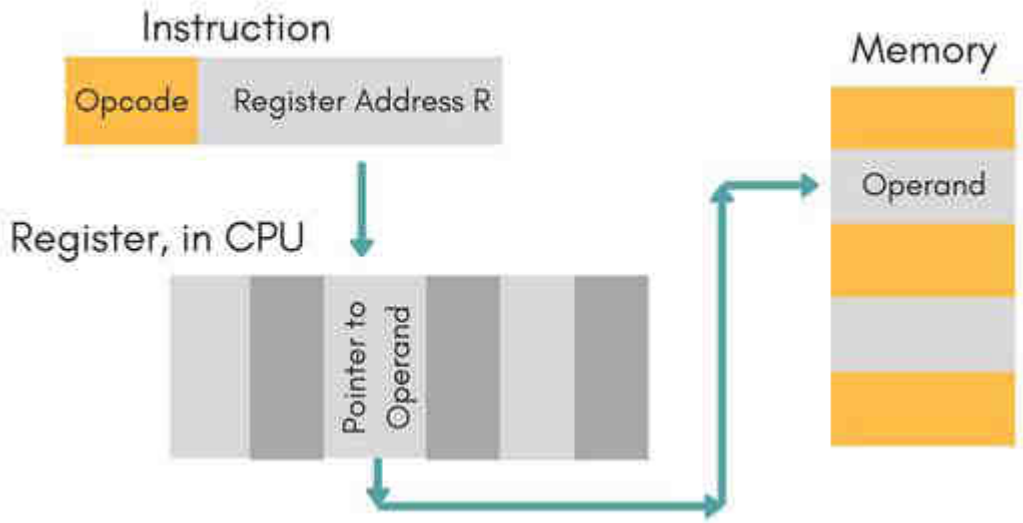
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

- Very limited address space
- Using multiple registers helps performance but it complicates the instructions.

Register Indirect Mode

In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.



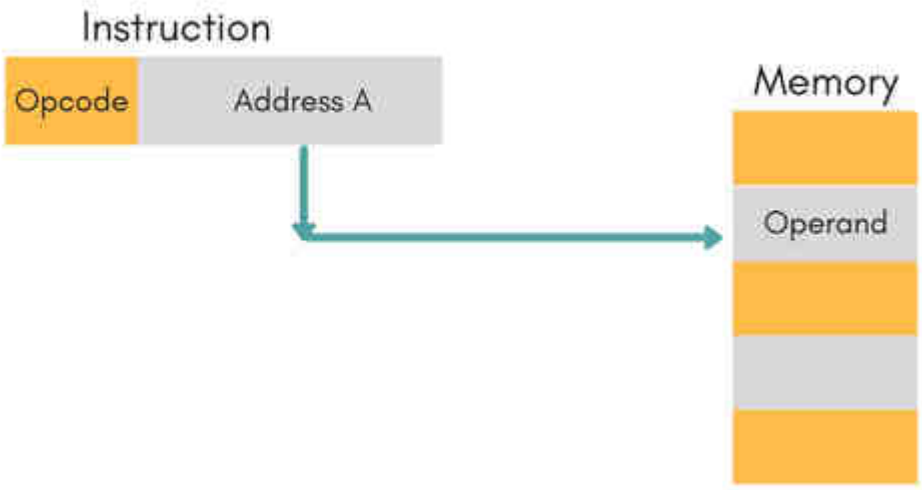
Auto Increment/Decrement Mode

In this the register is incremented or decremented after or before its value is used.

Direct Addressing Mode

In this mode, effective address of operand is present in instruction itself.

- Single memory reference to access data.
- No additional calculations to find the effective address of the operand.

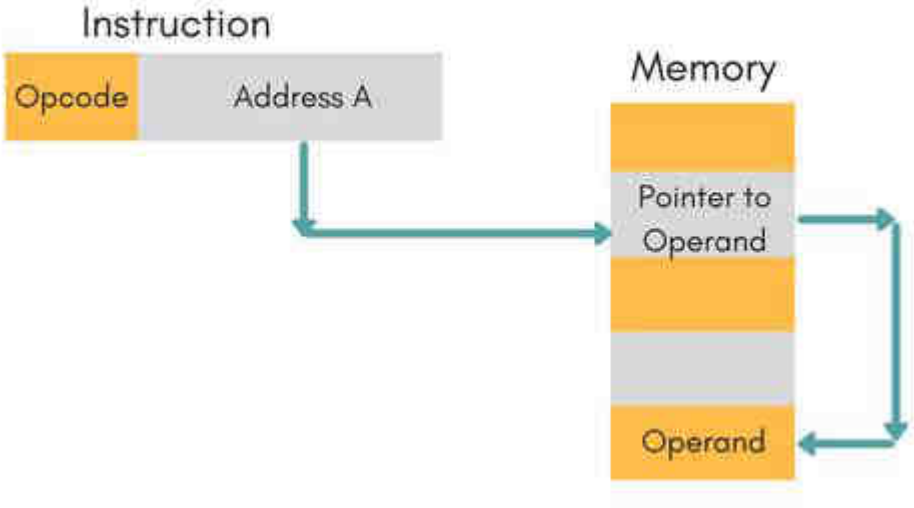


For Example: `ADD R1, 4000` - In this the 4000 is effective address of operand.

NOTE: Effective Address is the location where operand is present.

Indirect Addressing Mode

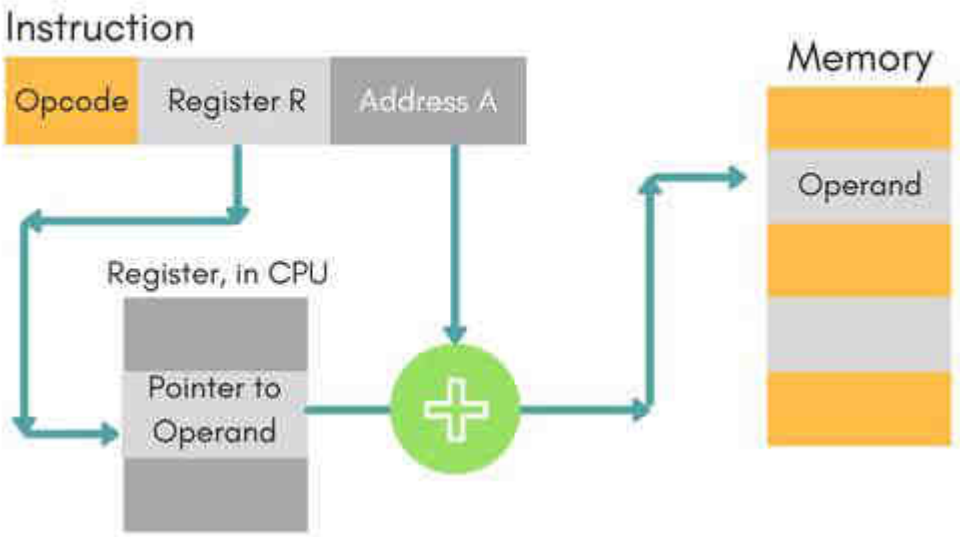
In this, the address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.



Displacement Addressing Mode

In this the contents of the indexed register is added to the Address part of the instruction, to obtain the effective address of operand.

`EA = A + (R)`, In this the address field holds two values, A(which is the base value) and R(that holds the displacement), or vice versa.



Relative Addressing Mode

It is a version of Displacement addressing mode.

In this the contents of PC(Program Counter) is added to address part of instruction to obtain the effective address.

`EA = A + (PC)`, where EA is effective address and PC is program counter.

The operand is A cells away from the current cell(the one pointed to by PC)

Base Register Addressing Mode

It is again a version of Displacement addressing mode. This can be defined as `EA = A + (R)`, where A is displacement and R holds pointer to base address.

Stack Addressing Mode

In this mode, operand is at the top of the stack. For example: `ADD`, this instruction will *POP* top two items from the stack, add them, and will then *PUSH* the result to the top of the stack.

Instruction Cycle

An instruction cycle, also known as **fetch-decode-execute cycle** is the basic operational process of a computer. This process is repeated continuously by CPU from boot up to shut down of computer.

Following are the steps that occur during an instruction cycle:

1. Fetch the Instruction

The instruction is fetched from memory address that is stored in PC(Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed.

2. Decode the Instruction

The instruction in the IR is executed by the decoder.

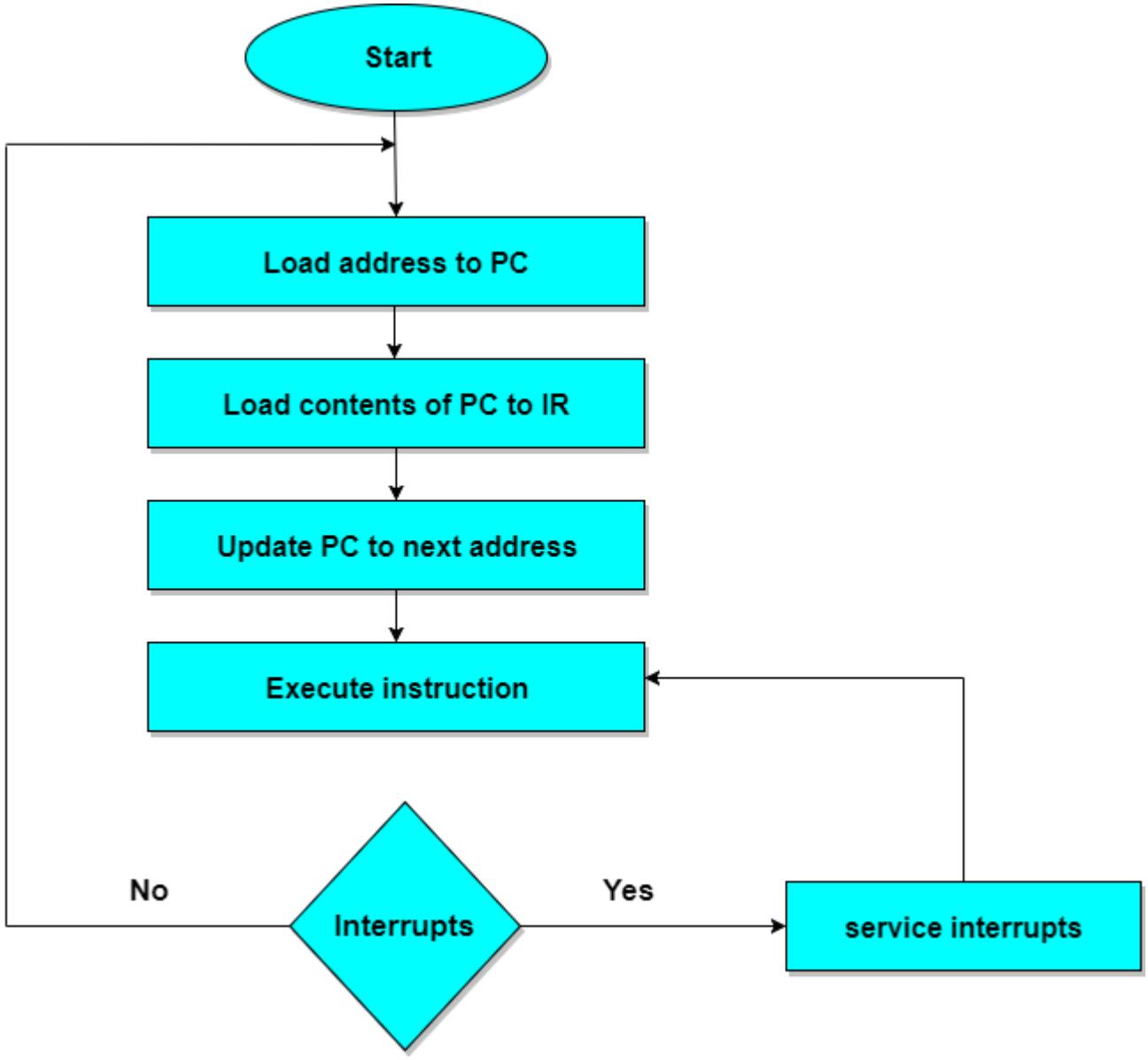
3. Read the Effective Address

If the instruction has an indirect address, the effective address is read from the memory. Otherwise operands are directly read in case of immediate operand instruction.

4. Execute the Instruction

The Control Unit passes the information in the form of control signals to the functional unit of CPU. The result generated is stored in main memory or sent to an output device.

The cycle is then repeated by fetching the next instruction. Thus in this way the instruction cycle is repeated continuously.


Start
Load address to PC
Load contents of PC to IR
Update PC to next address
Execute instruction
Interrupts
No
Yes
service interrupts

Computer Organization | Instruction Formats (Zero, One, Two and Three Address Instruction)

Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields. These field contains different information as for computers every thing is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:

- Operation field which specifies the operation to be performed like addition.
- Address field which contain the location of operand, i.e., register or memory location.
- Mode field which specifies how operand is to be founded.

A instruction is of various length depending upon the number of addresses it contain. Generally CPU organization are of three types on the basis of number of address fields:

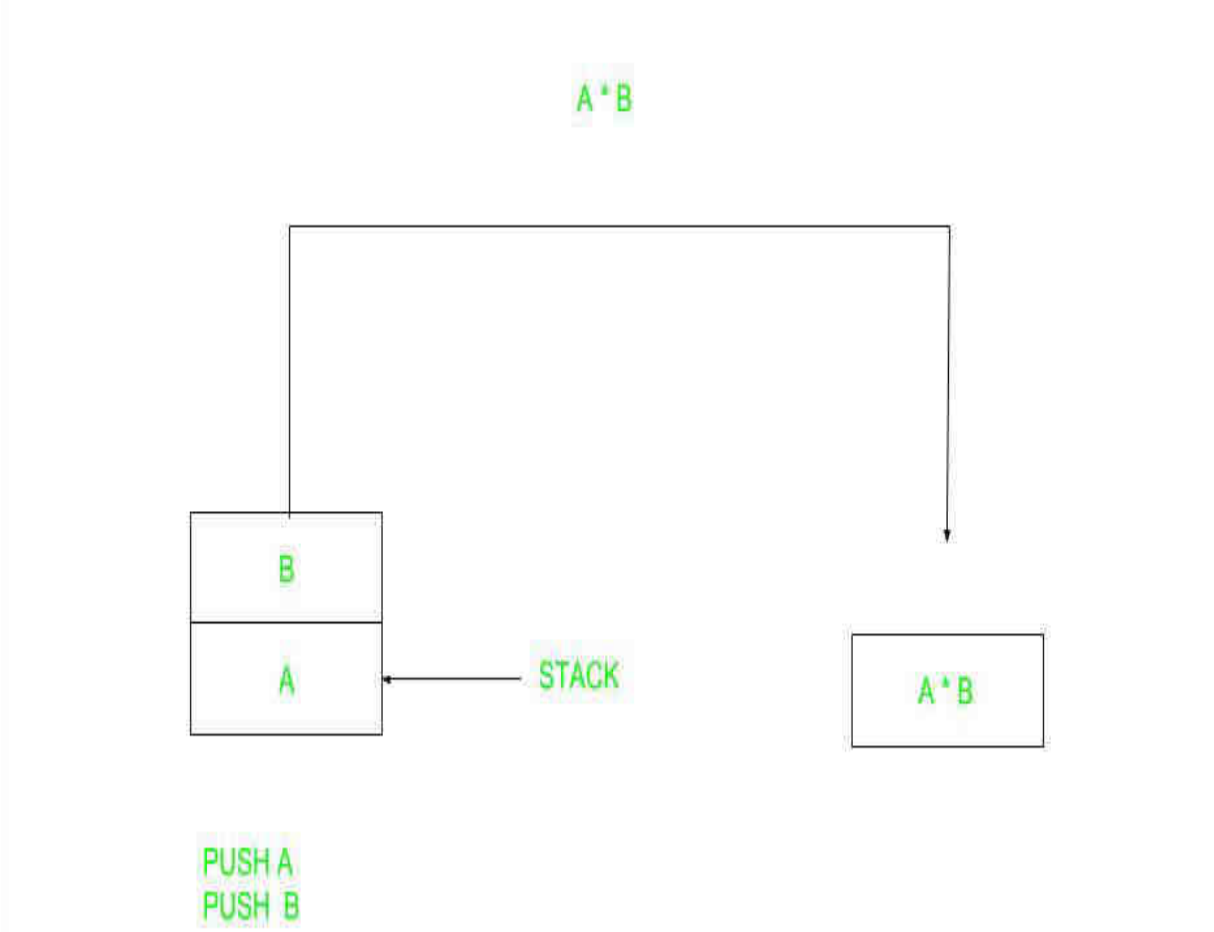
1. Single Accumulator organization
2. General register organization
3. Stack organization

In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is is applied a blend of various organization is mostly what we see generally.

On the basis of number of address instruction are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

1. **Zero Address Instructions –**

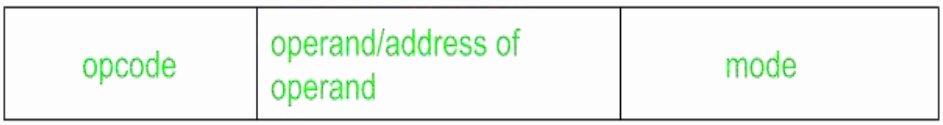


A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

```
Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location
```

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

2. **One Address Instructions –**
   This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.




```
Expression: X = (A+B)*(C+D)
AC is accumulator
M[] is any memory location
M[T] is temporary location
```

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

3. **Two Address Instructions –**
   This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

Here destination address can also contain operand.

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2

```
MOV     X, R1           M[X] = R1
```

4. **Three Address Instructions –**
   This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.


```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

```
ADD     R1, A, B        R1 = M[A] + M[B]
ADD     R2, C, D        R2 = M[C] + M[D]
MUL     X, R1, R2         M[X] = R1 * R2
```

Instruction Set 8085

1. Control
2. Logical
3. Branching
4. Arithmetic
5. Data Transfer

Control Instructions

Opcode	Operand	Explanation of Instruction	Description
NOP	**none**	No operation	No operation is performed. The instruction is fetched and decoded. However no operation is executed. `Example: NOP`
HLT	**none**	Halt and enter wait state	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. `Example: HLT`
DI	**none**	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. `Example: DI`
EI	**none**	Enable interrupts	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flipflop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). `Example: EI`

RIM	**none**	Read interrupt mas	This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations. `Example: RIM`
SIM	**none**	Set interrupt mask	This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows. `Example: SIM`

LOGICAL INSTRUCTIONS

Opcode	Operand	Explanation of Instruction	Description

CMP	**R**
	
	M
		Compare register or memory with accumulator	The contents of the operand (register or memory) are M compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:
			
			if (A) < (reg/mem): carry flag is set
			if (A) = (reg/mem): zero flag is set
			if (A) > (reg/mem): carry and zero flags are reset
			
			`Example: CMP B or CMP M`

CPI	**8-bit data**	Compare immediate with accumulator	The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
			
			if (A) < data: carry flag is set
			if (A) = data: zero flag is set
			if (A) > data: carry and zero flags are reset
			
			`Example: CPI 89H`

ANA	**R** **M**	Logical AND register or memory with accumulator	The contents of the accumulator are logically ANDed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is

			specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. `Example: ANA B or ANA M`
ANI	**8-bit data**	Logical AND immediate with accumulator	The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. `Example: ANI 86H`
XRA	**R** **M**	Exclusive OR register or memory with accumulator	The contents of the accumulator are Exclusive ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. `Example: XRA B or XRA M`
XRI	**8-bit data**	Exclusive OR immediate with accumulator	The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. `Example: XRI 86H`
ORA	**R** **M**	Logical OR register or memory with accumulator	The contents of the accumulator are logically ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. `Example: ORA B or ORA M`
ORI	**8-bit data**	Logical OR immediate with accumulator	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. `Example: ORI 86H`
RLC	**none**	Rotate accumulator left	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected. `Example: RLC`
RRC	**none**	Rotate accumulator right	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected. `Example: RRC`

RAL	**none**	Rotate accumulator left through carry	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected. `Example: RAL`
RAR	**none**	Rotate accumulator right through carry	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected. `Example: RAR`
CMA	**none**	Complement accumulator	The contents of the accumulator are complemented. No flags are affected. `Example: CMA`
CMC	**none**	Complement carry	The Carry flag is complemented. No other flags are affected. `Example: CMC`
STC	**none**	Set Carry	Set Carry `Example: STC`

BRANCHING INSTRUCTIONS

Opcode	Operand	Explanation of Instruction	Description
JMP	**16-bit address**	Jump unconditionally	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. `Example: JMP 2034H or JMP XYZ`
<table><tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr><tr><td>**JC**</td><td>**Jump on Carry**</td><td>**CY = 1**</td></tr></table>	**16-bit address**	Jump conditionally	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.

JNC	**Jump on no Carry**	**CY = 0**			Example: JZ 2034H or JZ XYZ
JP	**Jump on positive**	**S = 0**			
JM	**Jump on minus**	**S = 1**			
JZ	**Jump on zero**	**Z = 1**			
JNZ	**Jump on no zero**	**Z = 0**			
JPE	**Jump on parity even**	**P = 1**			
JPO	**Jump on parity odd**	**P = 0**			
Opcode	Description	Flag Status	**16-bit address**	Unconditional subroutine call	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. Example: CALL 2034H or CALL XYZ
CC	**Call on Carry**	**CY = 1**			
CNC	**Call on no Carry**	**CY = 0**			
CP	**Call on positive**	**S = 0**			
CM	**Call on minus**	**S = 1**			
CZ	**Call on zero**	**Z = 1**			
CNZ	**Call on no zero**	**Z = 0**			
CPE	**Call on parity even**	**P = 1**			
CPO	**Call on parity odd**	**P = 0**			
RET			**none**	Return from subroutine unconditionally	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter,and program execution begins at the new address. Example: RET
Opcode	Description	Flag Status	**none**	Return from subroutine conditionally	The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
RC	**Return on Carry**	**CY = 1**			
RNC	**Return on no Carry**	**CY = 0**			
RP	**Return on positive**	**S = 0**			

<table>
<tr>
<td>
<table>
<tr><td>RM</td><td>Return on minus</td><td>S = 1</td></tr>
<tr><td>RZ</td><td>Return on zero</td><td>Z = 1</td></tr>
<tr><td>RNZ</td><td>Return on no zero</td><td>Z = 0</td></tr>
<tr><td>RPE</td><td>Return on parity even</td><td>P = 1</td></tr>
<tr><td>RPO</td><td>Return on parity odd</td><td>P = 0</td></tr>
</table>
</td>
<td></td>
<td></td>
<td>Example: RZ</td>
</tr>
<tr>
<td>PCHL</td>
<td>none</td>
<td>Load program counter with HL contents</td>
<td>The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

Example: PCHL</td>
</tr>
<tr>
<td>RST</td>
<td>0-7</td>
<td>Restart</td>
<td>The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

<table>
<tr><th>Instruction</th><th>Restart Address</th></tr>
<tr><td>RST 0</td><td>0000H</td></tr>
<tr><td>RST1</td><td>0008H</td></tr>
<tr><td>RST 2</td><td>0010H</td></tr>
<tr><td>RST 3</td><td>0018H</td></tr>
<tr><td>RST 4</td><td>0020H</td></tr>
<tr><td>RST 5</td><td>0028H</td></tr>
<tr><td>RST 6</td><td>0030H</td></tr>
<tr><td>RST 7</td><td>0038H</td></tr>
</table>

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:</td>
</tr>
</table>

			<table><tr><th>Interrupt</th><th>Restart Address</th></tr><tr><td>TRAP</td><td>0024H</td></tr><tr><td>RST 5.5</td><td>002CH</td></tr><tr><td>RST 6.5</td><td>0034H</td></tr><tr><td>RST 7.5</td><td>003CH</td></tr></table>

Arithmetic Instructions

Opcode	Operand	Explanation of Instruction	Description
ADD	**R** **M**	Add register or memory, to accumulator	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. `Example: ADD B or ADD M`
ADC	**R** **M**	Add register to accumulator with carry	The contents of the operand (register or memory) and M the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. `Example: ADC B or ADC M`
ADI	**8-bit data**	Add immediate to accumulator	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. `Example: ADI 45H`
ACI	**8-bit data**	Add immediate to accumulator with carry	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

			Example: ACI 45H
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
DAD	**Reg. pair**	Add register pair to H and L registers	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H
SUB	**R** **M**	Subtract register or memory from accumulator	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SUB B or SUB M
SBB	**R** **M**	Subtract source and borrow from accumulator	The contents of the operand (register or memory) and M the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SBB B or SBB M
SUI	**8-bit data**	Subtract immediate from accumulator	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. Example: SUI 45H
SBI	**8-bit data**	Subtract immediate from accumulator with borrow	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example: XCHG
INR	**R** **M**	Increment register or memory by 1	The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: INR B or INR M
INX	**R**	Increment register pair by 1	The contents of the designated register pair are incremented by 1 and

			the result is stored in the same place. `Example: INX H`
DCR	**R** **M**	Decrement register or memory by 1	The contents of the designated register or memory are M decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. `Example: DCR B or DCR M`
DCX	**R**	Decrement register pair by 1	The contents of the designated register pair are decremented by 1 and the result is stored in the same place. `Example: DCX H`
DAA	**none**	Decimal adjust accumulator	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. `Example: DAA`

Opcode	Operand	Explanation of Instruction	Description
MOV	**Rd, Rs** **M, Rs** **Rd, M**	Copy from source(Rs) to destination(Rd)	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. `Example: MOV B, C or MOV B, M`
MVI	**Rd, data** **M, data**	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. `Example: MVI B, 57H or MVI M, 57H`
LDA	**16-bit address**	Load accumulator	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. `Example: LDA 2034H`
LDAX	**B/D Reg. pair**	Load accumulator indirect	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. `Example: LDAX B`
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designated in the operand. `Example: LXI H, 2034H or LXI H, XYZ`
LHLD	**16-bit address**	Load H and L registers direct	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. `Example: LHLD 2040H`
STA	**16-bit address**	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. `Example: STA 4350H`
STAX	**Reg. pair**	Store accumulator indirect	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. `Example: STAX B`
SHLD	**16-bit address**	Store H and L registers direct	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of

			H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. `Example: SHLD 2470H`
XCHG	**none**	Exchange H and L with D and E	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. `Example: XCHG`
SPHL	**none**	Copy H and L registers to the stack pointer	The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. `Example: SPHL`
XTHL	**none**	Exchange H and L with top of stack	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. `Example: XTHL`
PUSH	**Reg. pair**	Push register pair onto stack	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the highorder register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. `Example: PUSH B or PUSH A`
POP	**Reg. pair**	Pop off stack to register pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. `Example: POP H or POP A`
OUT	**8-bit port address**	Output data from accumulator to a port with 8-bit address	The contents of the accumulator are copied into the I/O port specified by the operand.

			Example: OUT F8H
IN	**8-bit port address**	Input data to accumulator from a port with 8-bit address	The contents of the input port designated in the operand are read and loaded into the accumulator. Example: IN 8CH

Data Transfer Instructions
