Chapter-2
Central Processing Unit-Part1

- Registers
- Addressing modes
- Instruction set
- Instruction Formats
- Instruction Types

Register

- A register is a very small amount of very fast memory that is built into the CPU (central processing unit).
- Contents can be accessed at extremely high speeds.
- Registers are used to store data temporarily during the execution of a program.
- Different processors have different register sizes.
- Registers are normally measured by the number of bits they can hold, for example, an 8-bit register means it can store 8 bits of data or a 32-bit register means it can store 32 bit of data

The transfer between the memory and the processor starts by sending the address of the memory location to the memory unit and issuing appropriate control signals

The memory address register (MAR) and memory data register (MDR) facilitates communication with memory

The MAR holds the address of the location to be accessed

The processor also contains numerous registers used for different purposes

The MDR contains the data to be written into or read out of the address location.

The program counter (PC) is a register that keeps track of the execution of a program – it contains the address of the next instruction to be fetch and executed

The instruction register (IR) holds the instruction that is currently being executed – its output to the control unit so the CU can generate the timing signals involved in executing the instruction

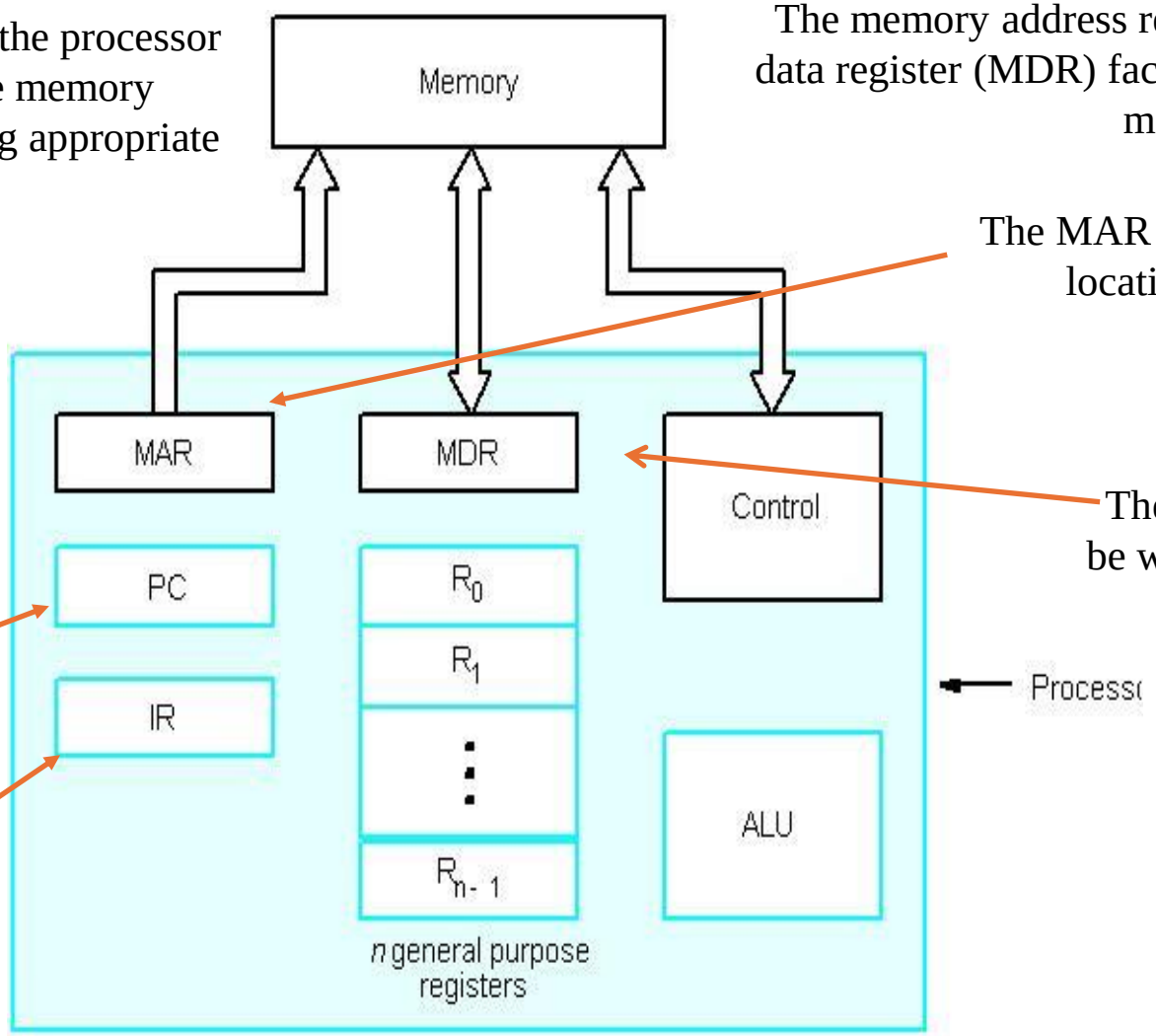


Figure 2.1 : Connection between the processor and memory

1.Program is loaded to memory via the input unit

2.Execution of the program starts when the program counter (PC) points to the first instruction

3.Contents of the PC is sent to the memory address register (MAR) and a read control signal is sent to memory

4.After memory access time finishes, the 1st instruction is read out of memory and loaded into the memory data register (MDR)

5.Contents of the MDR are transferred to the instruction register (IR)

6.Now the instruction is ready to be decoded and executed

7.Provided the instruction requires an operation that warrants the ALU, the operand is fetch from memory by sending the operand's address to the MAR and starting a Read cycle

8.Then operand is transferred from memory to the MDR

9.Then transferred from the MDR to the ALU

10.After all the operands are fetched – the ALU performs its operation

11.Result is stored in memory, then set to the MDR

12.The address of where the result will be stored in memory is sent to the MAR and a Write cycle is started

13.The PC is incremented to point to the next instruction

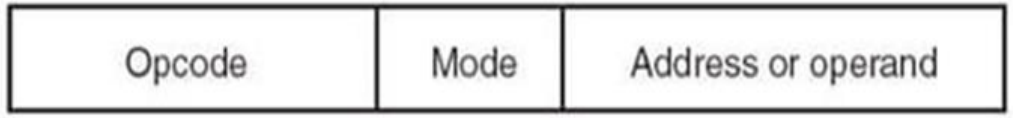
Instruction Formats

- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.
- The bits of the instruction are divided into groups called fields.
- The most common fields found in instruction formats are:

  1. An operation code field that specifies the operation to be performed

  2. An address field that designates a memory address or a processor register.

  3. A mode field that specifies the way the operand or the effective address is determined.



- Computers may have instructions of several different lengths containing varying numbers of addresses.
- The number of address fields in the instruct format of a computer depends on the internal organization of its registers.
- Most computers fall into one of three types of CPU organizations:
    1. Single accumulator organization.
    2. General register organization.
    3. Stack organization

General Register Organization

- The instruction format in this type of computer needs three register address fields.
- Thus, the instruction for an arithmetic addition may be written in an assembly language as
- **ADD R1, R2, R3** to denote the operation R1<-R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.
- The instruction **ADD R1, R2** would denote the operation R1<- *R1* + R2. Only register addresses for R1 and R2 need to be specified in this instruction.
- General register-type computers employ two or three address fields in their instruction format.
- Each address field may specify a processor register or a memory word.
- An instruction symbolized by **ADD R1, X** would specify the operation R1<- R1 + M[X].
- It has two address fields, one for register R1 and the other for the memory address X**.**

- The general register type computers employ two or three address fields in their instruction format. Each address field specifies a processor register or a memory. An instruction symbolized by ADD R1, X specifies the operation R1 ← R + M [X].
- This instruction has two address fields: register R1 and memory address X.

Stack Organization

- A computer with a stack organization has PUSH and POP instructions that require an address field. Hence, the instruction PUSH X pushes the word at address X to the top of the stack.
- The stack pointer updates automatically.
- In stack-organized computers, the operation type instructions don’t require an address field as the operation is performed on the two items on the top of the stack.

Single accumulator Organization

- The accumulator register is used implicitly for processing all instructions of a program and store the results into the accumulator.
- The instruction format that is used by this CPU Organisation is one address field. Due to this the CPU is known as One Address Machine.
- The main points about Single Accumulator based CPU Organisation is:

1. In this CPU Organization, the first ALU operand is always stored into the Accumulator and the second operand is present either in Registers or in the Memory.

2. Accumulator is the default address thus after data manipulation the results are stored into the accumulator.

3. One address instruction is used in this type of organization.

- The format of instruction is: Opcode + Address
- Opcode indicates the type of operation to be performed.

- Mainly two types of operation are performed in single accumulator based CPU organization:

i) Data transfer operation –

- In this type of operation, the data is transferred from a source to a destination.

For ex: LOAD X, STORE Y

- Here LOAD is memory read operation that is data is transfer from memory to accumulator and
- STORE is memory write operation that is data is transfer from accumulator to memory.

ii) ALU operation –

- In this type of operation, arithmetic operations are performed on the data.

For ex: MULT X

- where X is the address of the operand. The MULT instruction in this example performs the operation,
- AC <-- AC * M[X]
- AC is the Accumulator and M[X] is the memory word located at location X

Types of Instruction

- Depending on the multiple address fields, the instruction is categorized as follows:

1.Three address instruction

2.Two address instruction

3.One address instruction

4.Zero address instruction

1. Zero Address Instruction

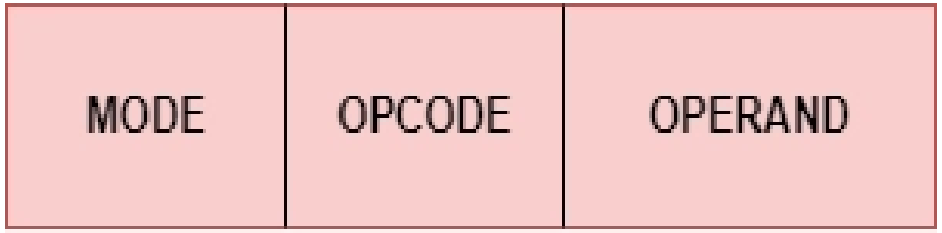
- This instruction does not have an operand field, and the location of operands is implicitly represented.
- The stack-organized computer system supports these instructions.
- To evaluate the arithmetic expression, it is required to convert it into reverse Polish notation.

Example: Consider the below operations, which shows how X = (A + B) ∗ (C + D) expression will be written for a stack-organized computer.

- TOS: Top of the Stack
- PUSH A TOS ← A
- PUSH B TOS ← B
- ADD TOS ← (A + B)
- PUSH C TOS ← C
- PUSH D TOS ← D
- ADD TOS ← (C + D)
- MUL TOS ← (C + D) ∗ (A + B)
- POP X M [X] ← TOS

2. One Address Instruction

- This instruction uses an implied accumulator for data manipulation operations.
- An accumulator is a register used by the CPU to perform logical operations.
- In one address instruction, the accumulator is implied, and hence, it does not require an explicit reference.
- For multiplication and division, there is a need for a second register. However, here we will neglect the second register and assume that the accumulator contains the result of all the operations.



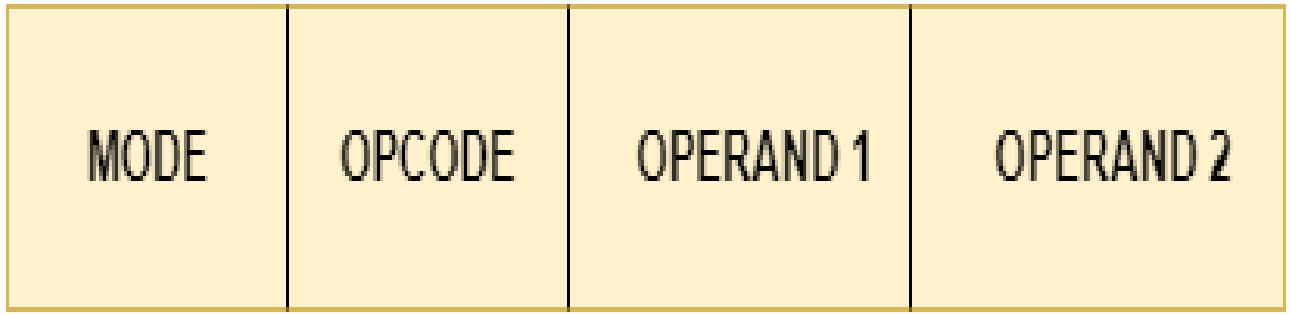
Example: The program to evaluate X = (A + B) ∗ (C + D) is as follows:

• LOAD	A	AC ← M [A]
• ADD	B	AC ← A [C] + 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

- All operations are done between the accumulator(AC) register and a memory operand.
- M[ ] is any memory location.
- M[T] addresses a temporary memory location for storing the intermediate result.
- This instruction format has only one operand field. This address field uses two special instructions to perform data transfer, namely:
- LOAD: This is used to transfer the data to the accumulator.
- STORE: This is used to move the data from the accumulator to the memory.

3. Two Address Instructions

- This instruction is most commonly used in commercial computers.
- This address instruction format has three operand fields.
- The two address fields can either be memory addresses or registers.



Example: The program to evaluate X = (A + B) ∗ (C + D) is as follows:

- MOV R1, A R1 ← M [A]
- ADD R1, B R1 ← R1 + M [B]
- MOV R2, C R2 ← M [C]
- ADD R2, D R2 ← R2 + M [D]
- MUL R1, R2 R1 ← R1∗R2
- MOV X, R1 M [X] ← R1

The MOV instruction transfers the operands to the memory from the processor registers. R1, R2 registers.

4. Three Address Instruction

- The format of a three-address instruction requires three operand fields. These three fields can be either memory addresses or registers.

Example: The program in assembly language X = (A + B) ∗ (C + D) Consider the instructions given below that explain each instruction's register transfer operation.

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

Two processor registers, R1 and R2.
The symbol M [A] denotes the operand at the memory address symbolized by A. The operand1 and operand2 contain the data or address that the CPU will operate. Operand 3 contains the result's address.

Instruction Set

- An **instruction set** in computer organization refers to the collection of instructions that a computer's central processing unit (CPU) can execute.
- The instructions are used to perform various operations such as data transfer, arithmetic, logic, control, and more.
- The design and architecture of an instruction set are fundamental to how a computer functions and processes information.

Types of Instructions in an Instruction Set

1. Data Transfer Instructions:

- These instructions are used to move data between registers, memory locations, and input/output devices.

2. Arithmetic Instructions:

- Perform basic mathematical operations like addition, subtraction, multiplication, and division.

3. Logical Instructions:

Perform logical operations like AND, OR, XOR, and NOT.

4. Control Flow Instructions:

- Control the sequence of execution of instructions by making decisions or altering the flow.

5. Input/Output Instructions:

- Handle data exchange between the CPU and external devices like keyboards, displays, and storage.

6. Shift and Rotate Instructions:

- Shift and rotate bits within a register.

7. Flag Control Instructions:

- Modify the status flags used by the CPU for conditional operations.

Data- transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOVE
Exchange	XCH
Push	PUSH
Pop	POP
Input	IN
Output	OUT

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Subtract reverse	SUBR
Negate	NEG

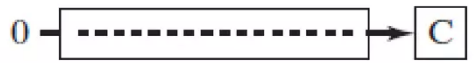
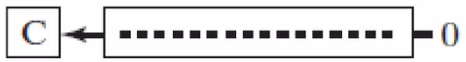
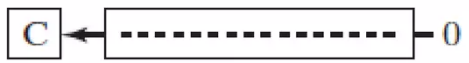
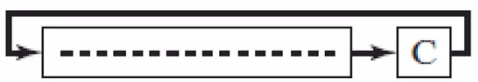
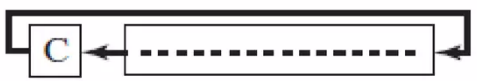
Logical Instructions

Name	Mnemonic
Clear	CLR
Set	SET
Complement	NOT
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC

Control Flow Instructions

Name	Mnemonic
Unconditional Jump	JMP
Call a Subroutine	CALL
Return from Subroutine	RET
Conditional Branch Instructions	JZ (Jump if Zero)
	JNZ (Jump if Not Zero)
	JC (Jump if Carry)
	JNC (Jump if No Carry)
	JE (Jump if Equal)
	JNE (Jump if Not Equal)
LOOP Instruction	LOOP
Interrupt	INT

Shift and Rotate Instructions:

Name	Mnemonic	Diagram
Logical shift right	SHR	
Logical shift left	SHL	
Arithmetic shift right	SHRA	
Arithmetic shift left	SHLA	
Rotate right	ROR	
Rotate left	ROL	
Rotate right with carry	RORC	
Rotate left with carry	ROLC	

IN and OUT Instructions

Name	Mnemonic
IN Instruction	IN
OUT Instruction	OUT

Syntax: IN destination_register, port_address

Syntax: OUT port_address, source_register

Flag Control Instructions

Name	Mnemonic
Set Carry Flag	STC
Clear Carry Flag	CLC
Complement Carry Flag	CMC
Clear Direction Flag	CLD
Set Direction Flag	STD

Addressing mode

- The different ways in which the location of an operand is specified in an instruction are referred to as Addressing Modes

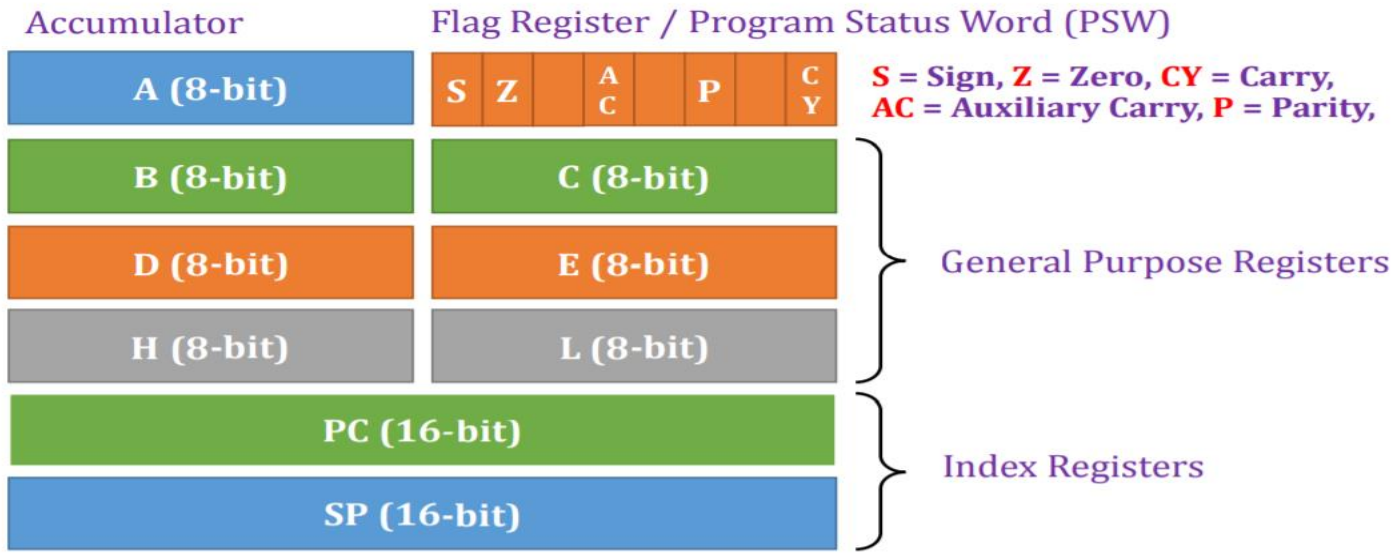
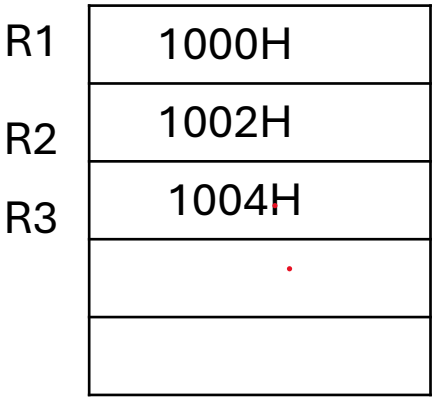
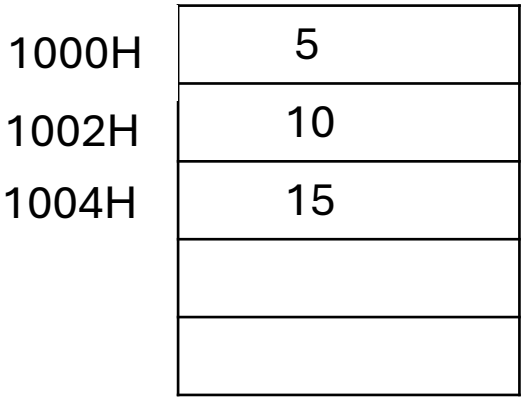


Fig: Register inside the CPU

***Effective Address (EA): EA is the address of the exact memory location where the value of the operand is present





Processor register

Memory register

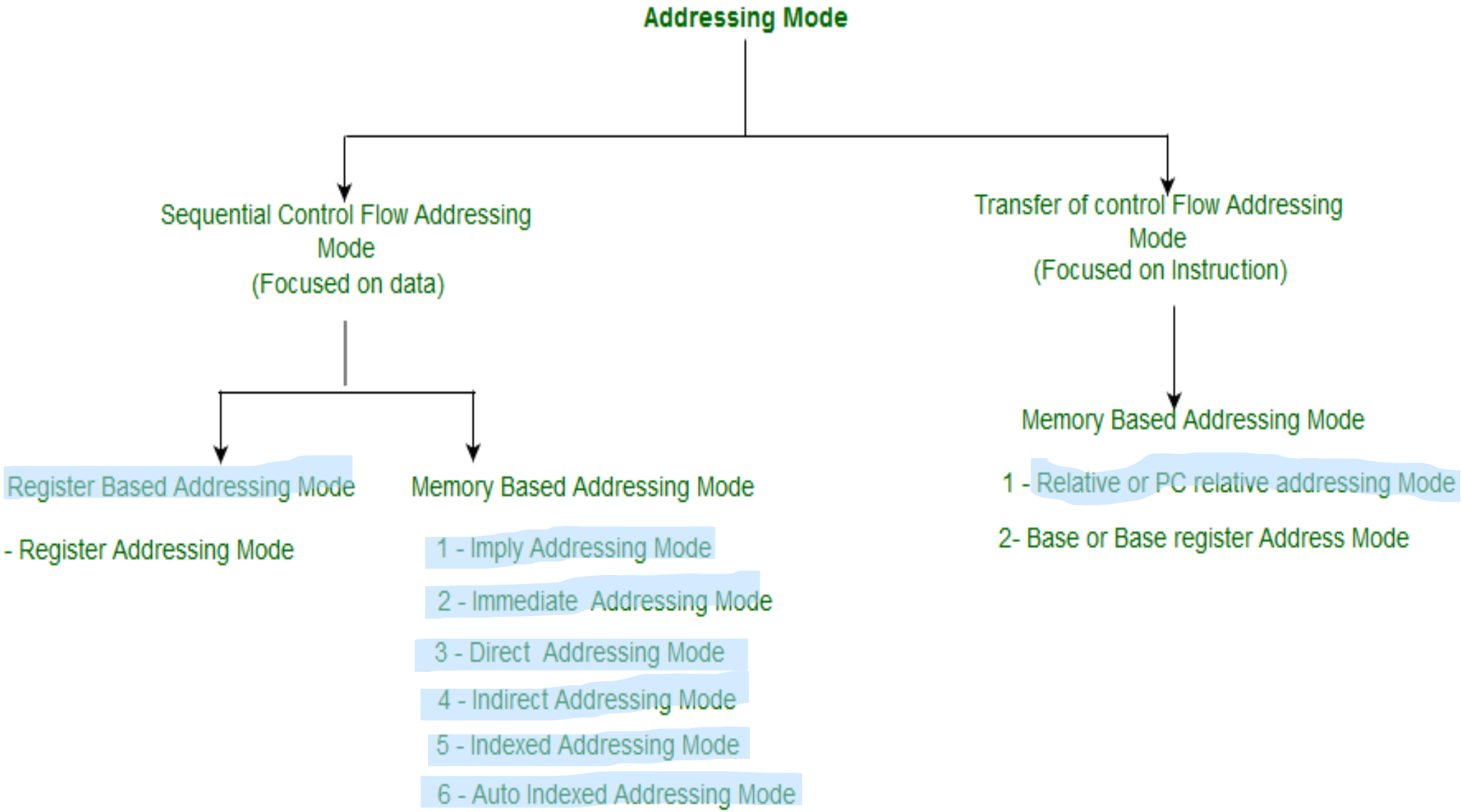
Handle data and control flow information with the CPU for quick access and computation.

Deals with the address and data in the memory

- Microprocessor executes the instructions stored in memory (RAM).
- It executes one instruction at a time.
- Each of the instruction contains operations and operands.
- Operation specifies the type of action to be performed.
- For example: ADD, SUB, MOV, INC, LOAD, STORE
- Operands are the data on which the operation is to be performed.

  MOV B, A (Here MOV is operation and (B & A) are operands)

  ADD B (Here ADD is operation and (B) is operand)
- Operand can be place either in one of the processor register or in memory.
- There are different ways to get the operands.
- The way in which the operand is taken from register or memory is named as addressing mode


Addressing Mode
Sequential Control Flow Addressing Mode
(Focused on data)
Transfer of control Flow Addressing Mode
(Focused on Instruction)
Register Based Addressing Mode
- Register Addressing Mode
Memory Based Addressing Mode
1 - Imply Addressing Mode
2 - Immediate Addressing Mode
3 - Direct Addressing Mode
4 - Indirect Addressing Mode
5 - Indexed Addressing Mode
6 - Auto Indexed Addressing Mode
Memory Based Addressing Mode
1 - Relative or PC relative addressing Mode
2- Base or Base register Address Mode

1. Implied addressing mode

- It is also called implicit/inherent addressing mode.
- The operand is implied by the instruction.
- The operand is hidden/fixed inside the instruction.

Example:

Complement Accumulator - CMA
(Here accumulator A is implied by the instruction)
Complement Carry Flag - CMC
(Here Flags register is implied by the instruction)
Set Carry Flag - STC
(Here Flags register is implied by the instruction)

2. Immediate Addressing Mode

The operand is specified with in the instruction.

- Operand itself is provided in the instruction rather than its address. Example:

Move Immediate

MVI A , 15h A ← 15h

(Here 15h is the immediate operand)

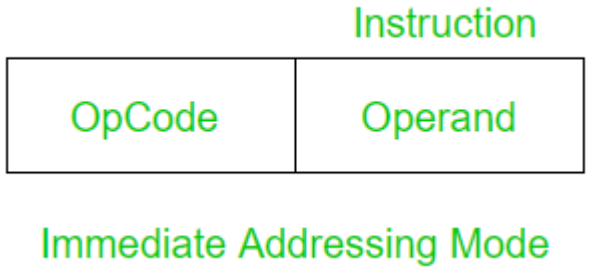


Figure: Immediate Addressing Diagram

3. Direct Addressing mode:

- The instruction specifies the direct address of the operand.
- The memory address is specified where the actual operand is.

Example:

Load Accumulator

LDA 2805h A ← [2805]

(It loads the data from memory location 2805 to A)

Store Accumulator

STA 2803h [2803] ← A

(It stores the data from A to memory location 2803)

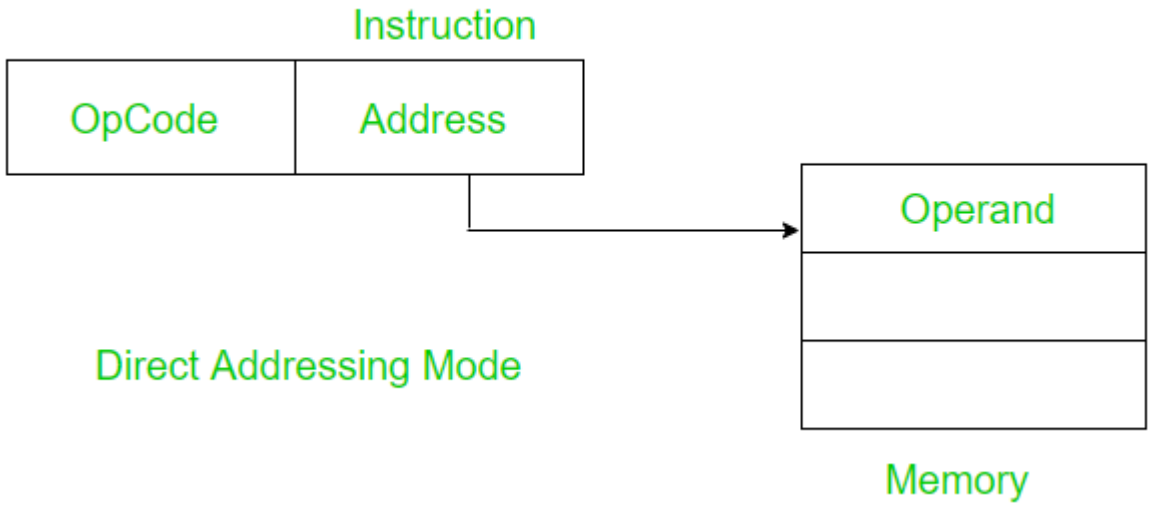


Figure: Direct Addressing Diagram

- **Effective address (EA) = address field (A)**

e.g. ADD A ;
Add contents of cell A to accumulator

4. Indirect Addressing mode:

- The instruction specifies the indirect address where the effective address of the operand is placed.
- The memory address is specified where the actual address of operand is placed.

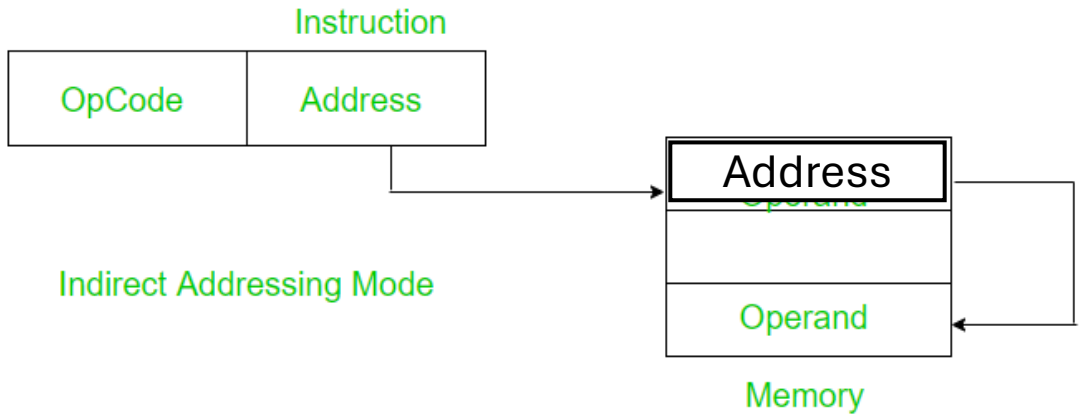


Figure: Indirect Addressing Diagram

Example:

Move

MOV A, 2802h A ← [[2802]]

It moves the data from memory location specified by the location 2802 to A

There are 2 types(or versions) of Indirect Addressing Mode: Memory Indirect, and Register Indirect.

1. Memory Indirect – In this type we directly mention the address of the memory location in the instruction either enclosed by parenthesis or preceded by '@' character.
Example : LOAD R1, (1005) or LOAD R1, @1005

2. Register Indirect – In this type the address of the target memory location will be stored in the register and the register will be mentioned in the instruction.
Example: MOV R@, 1005
LOAD R1, (R2)

5.Indexed Addressing mode

In index addressing mode, contents of Index register is added to address part of instruction to obtain effective address.

- The address part of instruction holds the beginning/base address and is called as base.
- The index register hold the index value, which is +ve.
- Base remains same, the index changes.
- When the base is added to the index register the resultant number is the memory location where the operand will be placed.

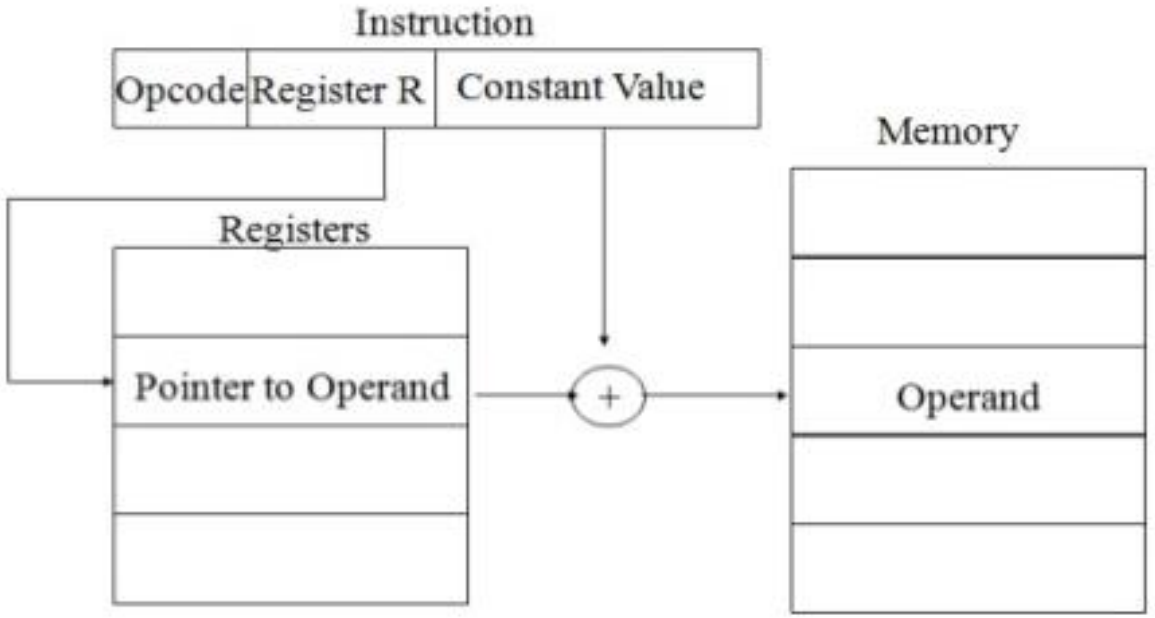


Fig: Indexed Addressing mode

- EA = X + [R]

Address field hold two values : X = constant value (offset)

R = register that holds address of memory locations

6. Auto Indexed addressing mode:

1. Auto Indexed (increment mode):

- Effective address of the operand is the contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.

(R1)+.

Here one register reference,one memory reference and one ALU operation is required to access the data.

Example: *Add R1, (R2)+ // OR*

$R1 = R1 + M[R2]$

$R2 = R2 + d$

Useful for stepping through arrays in a loop. R2 – start of array d – size of an element

2. Auto indexed (decrement mode):

- Effective address of the operand is the contents of a register specified in the instruction.
- Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location.

–(R1)

Here one register reference, one memory reference and one ALU operation is required to access the data.

Example:

Add R1,-(R2) //OR

R2 = R2-*d*

R1 = R1 + M[R2]

Note: Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop . Auto increment and Auto decrement modes are useful for implementing "Last-In-First-Out" data structures

7. Relative Addressing Mode:

- In the case of relative addressing mode, we need to add a constant to the content of the register in order to refer to the address of the next operand.
- But in some computers, the program counter is used instead of using a register
- Here is a symbolic representation of the relative address mode:

  X(PC)

- Here is how the effective address for it:

  EA = X + (PC)

- Since the operand addresses here are found relative to a program counter, it is known as the relative address mode.

Example:

1000H: Add R1, R2

1002H: JUMP LABEL

1004H: Sub R3, R4

:

:

1010H: LABEL: MUL R5, R6

8. Register Addressing Mode

- Register addressing mode is a commonly used addressing mode.
- In this mode, the operands of an instruction get specified by referring to registers.
- The instruction operates on values stored in registers.
- Effective Address= R
- Example: Add R4, R3

  Load R3, R2