Computer Organization and Architecture

FOR CSE /CST, 4TH SEM

PRESENTD BY : JEMARANI JAYPURIA

SYLLABUS

- **UNIT:1**

FUNDAMENTALS OF A COMPUTER SYSTEM: Functional Units of a Digital Computer ,Hardware ,Software Interface, Translation from a High Level Language to the Hardware Language Instruction Set Architecture, Styles and features, RISC and CISC Architectures ,Performance Metrics ,Amdahl's Law ,Case Studies of ISA.

UNIT:2

BASIC PROCESSING UNIT: Components of the Processor, Data path and Control – Execution of a Complete Instruction, Hardwired and Micro programmed Control, Instruction Level Parallelism, Basic Concepts of Pipelining, Pipelined Implementation of Data path and Control, Hazards, Structural, Data and Control Hazards ,Exception handling. Parallelism and Multiprocessor Architecture, Flynn's Classification, UMA, NUMA, Distributed Memory Architecture. Array and Vector Processor.

SYLLABUS

- **UNIT:3**

ARITHMETIC FOR COMPUTERS: Addition and Subtraction, Fast Adders, Binary Multiplication, Fast Multiplication, Binary Division and its techniques, Floating Point Numbers, Representation, Arithmetic Operations.

- **UNIT:4**

MEMORY AND I/O : Need for a hierarchical memory system, Types and characteristics of memories ,Memory location and address, Endianness of memory representation, Cache memories, Improving cache performance, Virtual memory ,Memory management techniques, cache mapping and its techniques ,Associative memories. Page Replacement Algorithms. Accessing I/O devices – Programmed Input/output, Interrupts, Direct Memory Access ,Interface circuits ,Need for Standard I/O Interfaces like PCI, SCSI, USB.

Books

Text Books:

1. Carl Hamacher, ZvonkoVranesic, SafwatZaky and NaraigManjikian, "Computer Organization and Embedded Systems", Sixth Edition, Tata McGraw Hill, 2012.
2. David A. Patterson and John L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface", Fourth Edition, Morgan Kaufmann / Elsevier, 2009.
3. Kai Hwang and F.A. Briggs, "Computer Architecture and Parallel Processing", McGraw Hill.

Reference Books:

4. M. Morris Mano, "Computer System Architecture", PHI
5. William Stallings, "Computer Organization and Architecture – Designingfor Performance", Sixth Edition, Pearson Education, 2003.
6. John P. Hayes, "Computer Architecture and Organization", Third Edition, Tata McGraw Hill, 1998.
7. John L. Hennessey and David A. Patterson, "Computer Architecture – A Quantitative Approach", Morgan Kaufmann / Elsevier Publishers, Fifth Edition, 2012.

Computer Architecture	Computer Organization
Architecture describes what the computer does.	**The Organization describes how it does it.**
Computer Architecture deals with the functional behavior of computer systems.	**Computer Organization deals with a structural relationship.**
It deals with high-level design issues.	**It deals with low-level design issues.**
Architecture indicates its hardware.	**Where Organization indicates its performance.**
For designing a computer, its architecture is fixed first.	**For designing a computer, an organization is decided after its architecture.**
Computer Architecture is also called instruction set architecture.	**Computer Organization is frequently called microarchitecture.**
Computer Architecture comprises logical functions such as instruction sets, registers, data types, and addressing modes.	**Computer Organization consists of physical units like circuit designs, peripherals, and adders.**
Architecture coordinates between the hardware and software of the system.	**Computer Organization handles the segments of the network in a system.**

FUNDAMENTALS OF A COMPUTER SYSTEM

A typical digital computer system has four basic functional elements

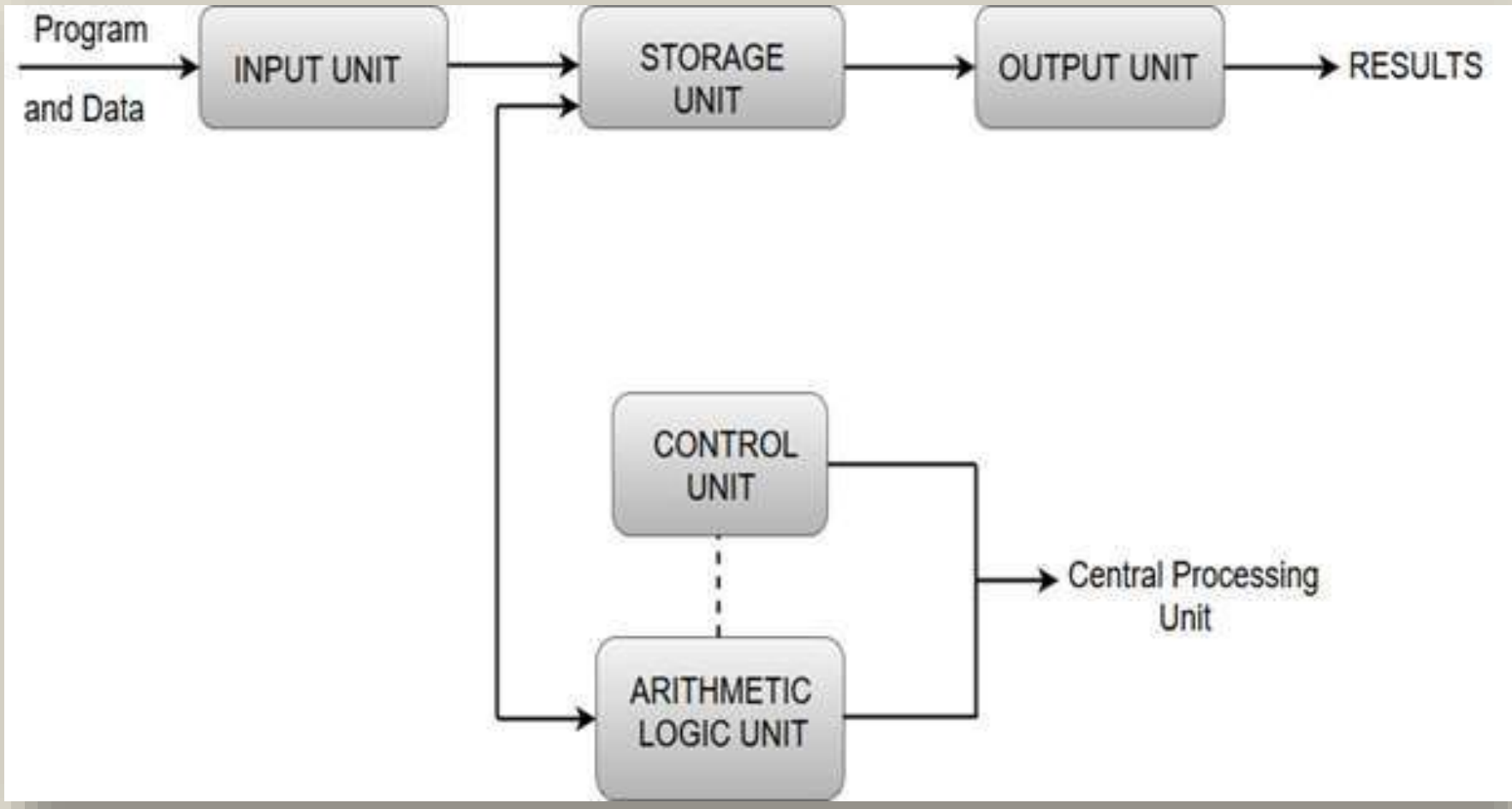
(1) Input-output equipment,

(2) Main memory,

(3) Control unit,

(4) Arithmetic-logic unit.

Functional Units of Digital System



Input unit

- Input units are used by the computer to read the data.

e.g. keyboards, mouse, joysticks, trackballs, microphones, etc

Trackballs

Joysticks

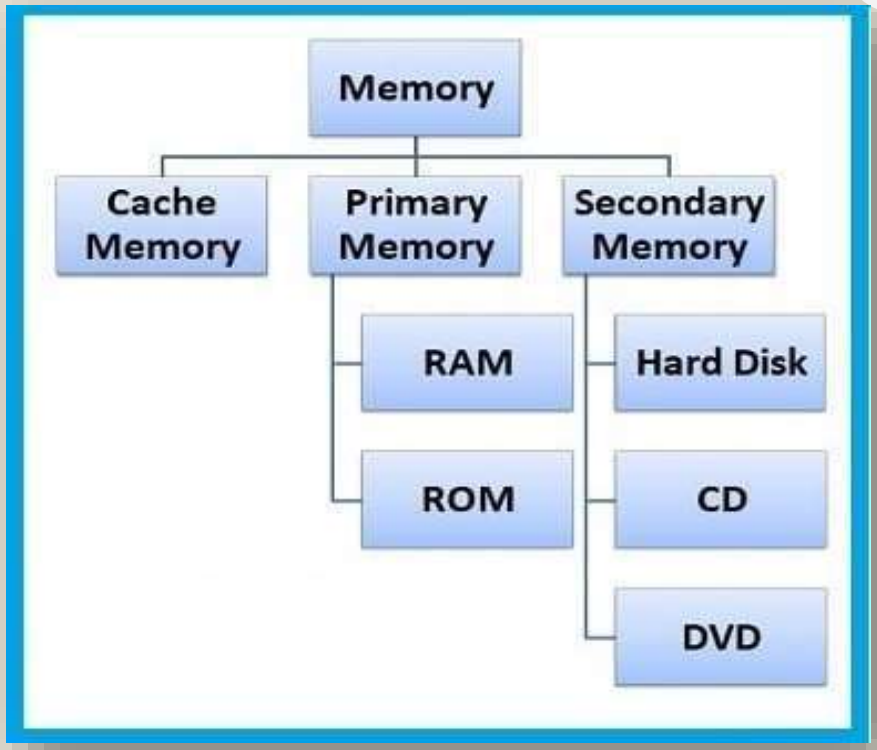
Input devices

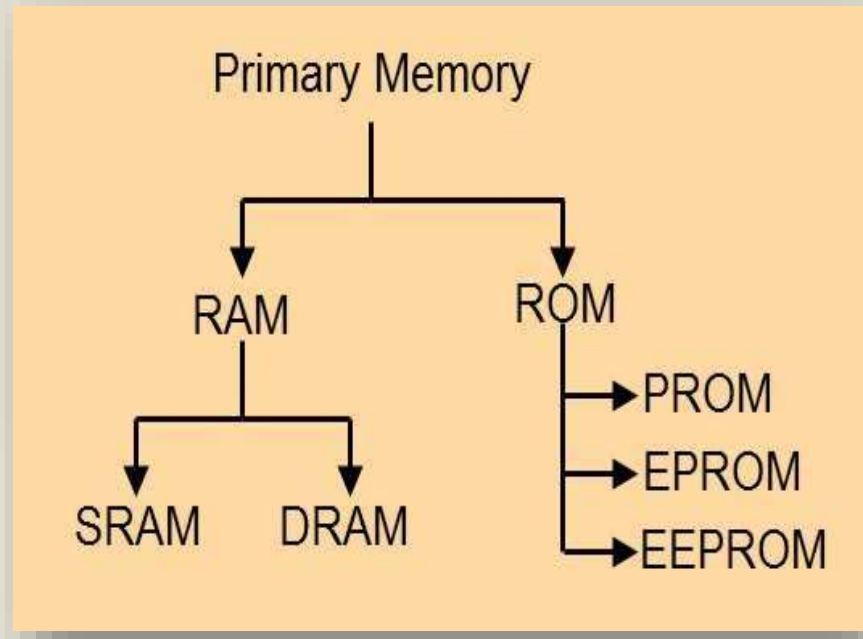
Central processing unit

- It carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Memory unit

- Programs are kept which are running, and that contains data needed by the running programs
- Primary memory and secondary memory.
- It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.


Memory
Cache Memory
Primary Memory
Secondary Memory
RAM
ROM
Hard Disk
CD
DVD


Primary Memory
RAM
ROM
SRAM
DRAM
PROM
EPROM
EEPROM

Primary storage

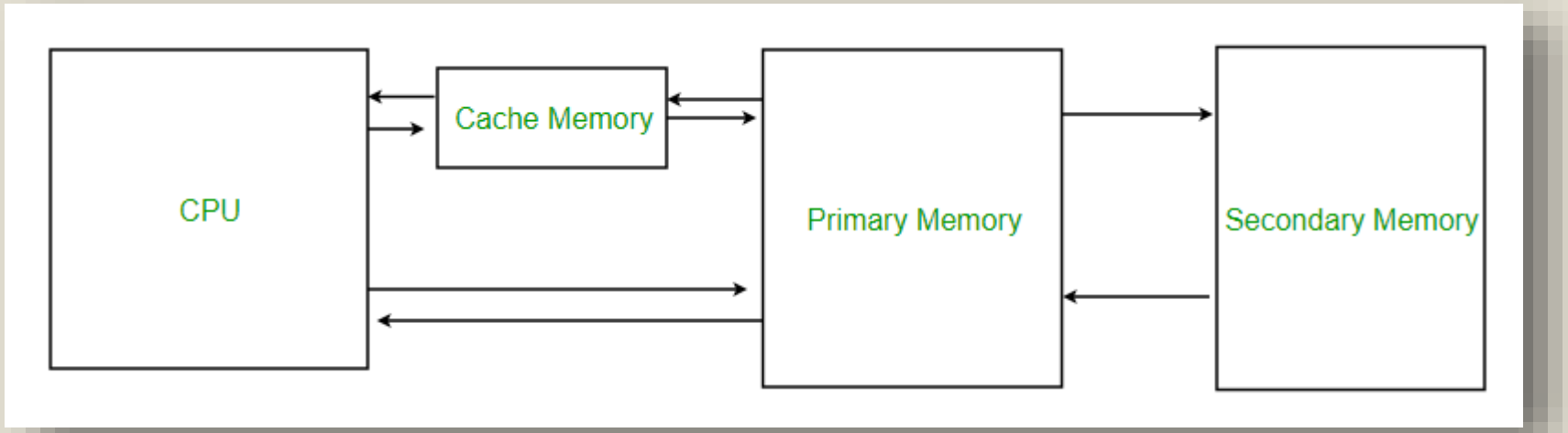
- Fastest memory that operates at electronic speeds
- Primary memory contains a large number of semiconductor storage cells, capable of storing a bit of information. The word length of a computer is between 16-64 bits.
- It is also known as the volatile form of memory, means when the computer is shut down, anything contained in RAM is lost.
- Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor
- The most common examples of primary memory are RAM and ROM

SRAM vs DRAM

VS

Difference Between SRAM & DRAM

Cache memory



Cache Memory

- As an adjunct to the main memory, a smaller, faster RAM unit, called a cache, is used to hold sections of a program that are currently being executed, along with any associated data.
- The purpose of the cache is to facilitate high instruction execution rates.
- When the execution of an instruction requires data, located in the main memory, the data are fetched and copies are also placed in the cache. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use.
- The cache is smaller in size. Memory ranges from **2KB to a few MB** generally.

Secondary memory

- Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.
- It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.
- E.g . Magnetic disks, magnetic tapes, and optical disks.

Arithmetic & logical unit

- Most of all the arithmetic and logical operations of a computer are executed by ALU
- e.g.

Arithmetic operations like addition, subtraction, multiplication, division.

And also the logical operations like AND, OR, NOT operations.

Control unit

- Coordinates the operation of the processor
- It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.

e.g. Addition of two operands

"Add LOC A, RO"

This instruction adds the memory location LOC A to the operand in the register RO and places the sum in the register RO. This instruction internally performs several steps.

Basic Operational Concepts

- An Instruction consists of two parts, an Operation code and operand/s as shown below

OPCODE	OPERAND/s

Let us see a typical instruction ADD LOC A, Ro.

Step 1: Fetch the instruction from main memory into the processor.

Step 2: Fetch the operand at location LOC A from main memory into the processor.

Step 3: Add the memory operand (i.e. fetched contents of LOCA) to the contents of register Ro.

Step 4: Store the result (sum) in Ro.

Cont…

The same instruction can be realized using two instructions as

- Load LOC A,R1
- Add R1,Ro

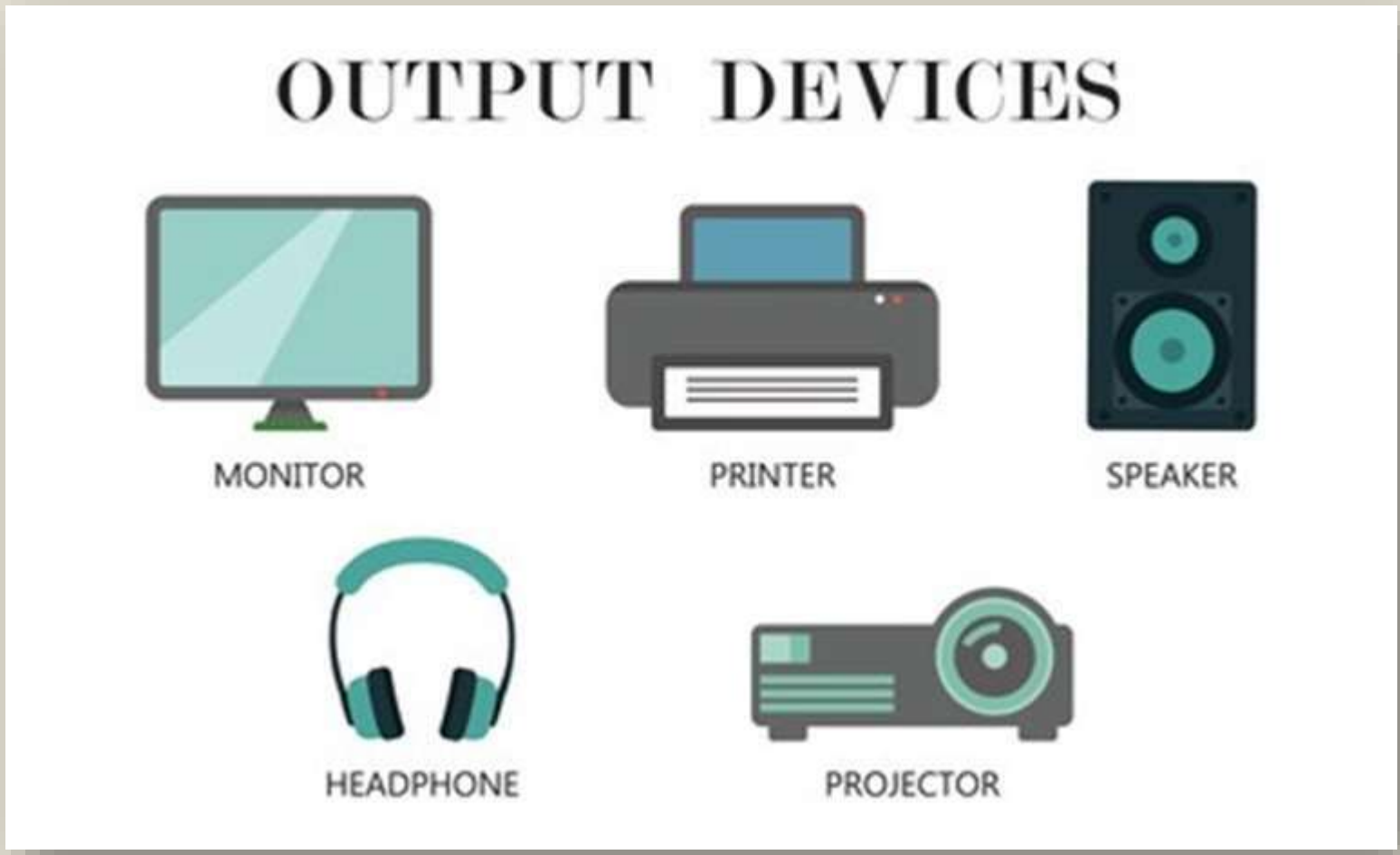
- Step 1: Fetch the instruction from main memory into the processor

- Step 2: Fetch the operand at location LOC A from main memory into the processor Register R1
- Step 3: Add the content of Register R1 and the contents of register
- Ro Step 4: Store the result (sum) in Ro.

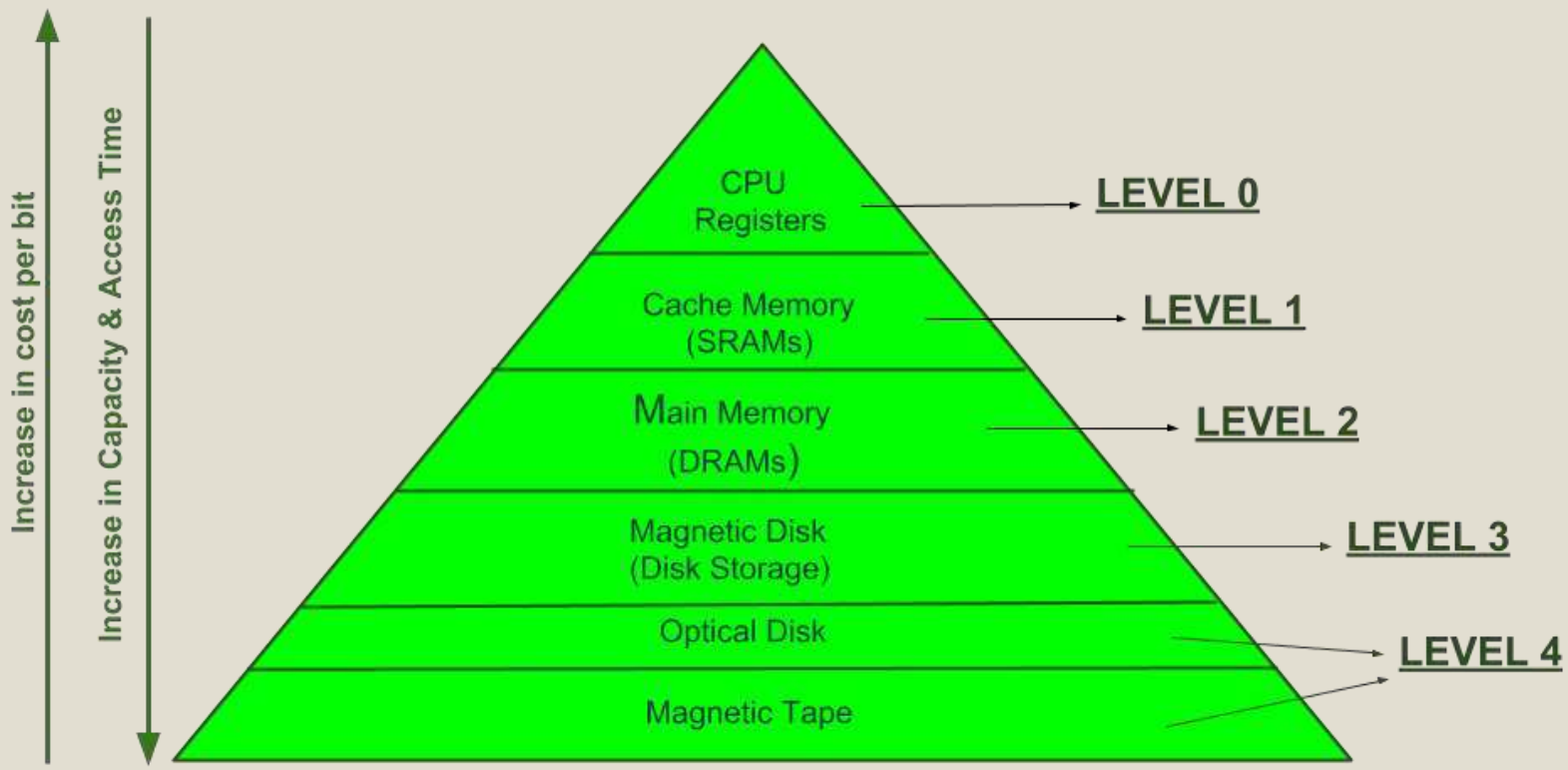
Output Unit

- The primary function of the output unit is to send the processed results to the user.
- Output devices are pieces of equipment that are used to generate information or any other response processed by the computer.
- The most common example of an output device is a monitor.

Output Devices



Memory Hierarchy Design



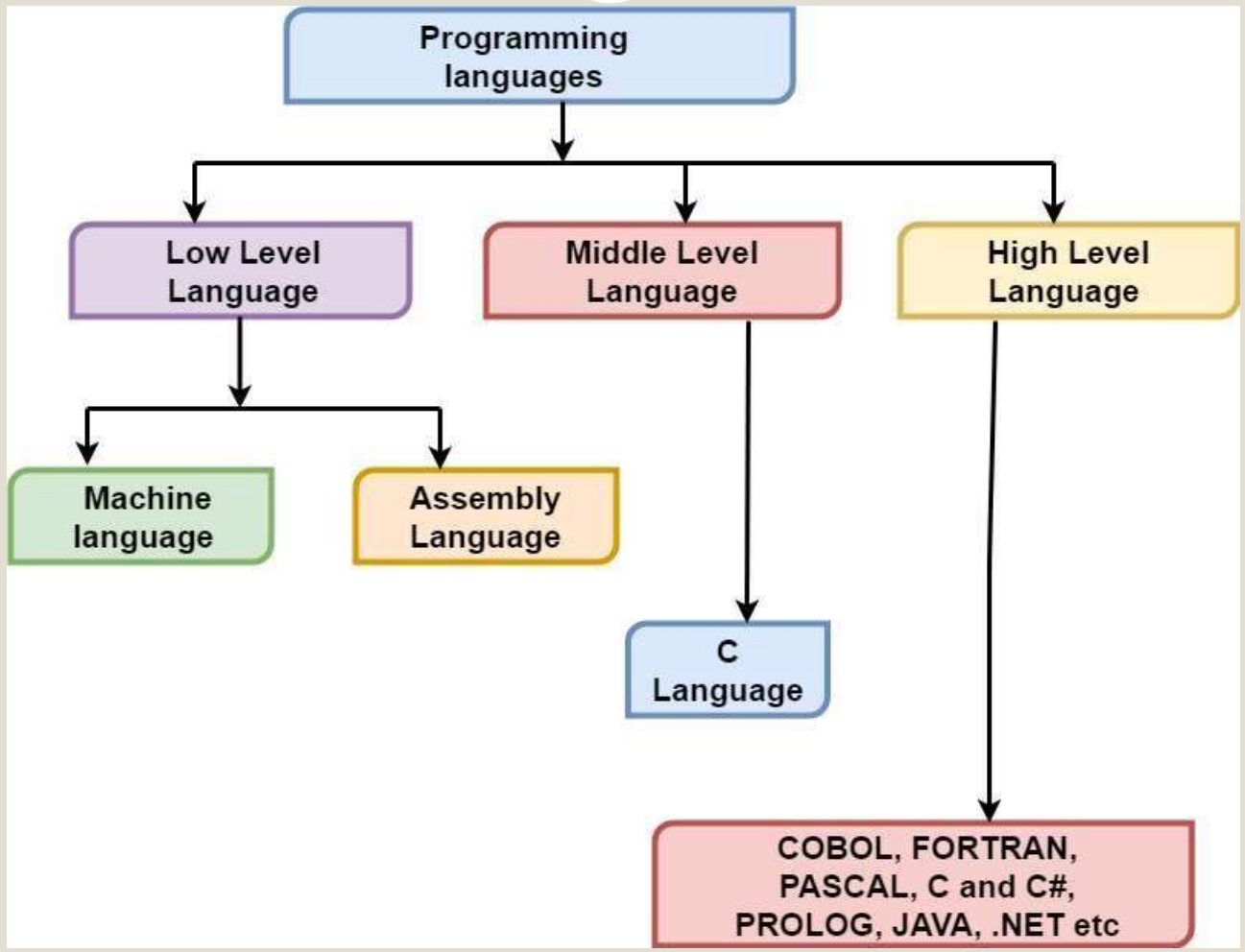
MEMORY HIERARCHY DESIGN

Qsn .

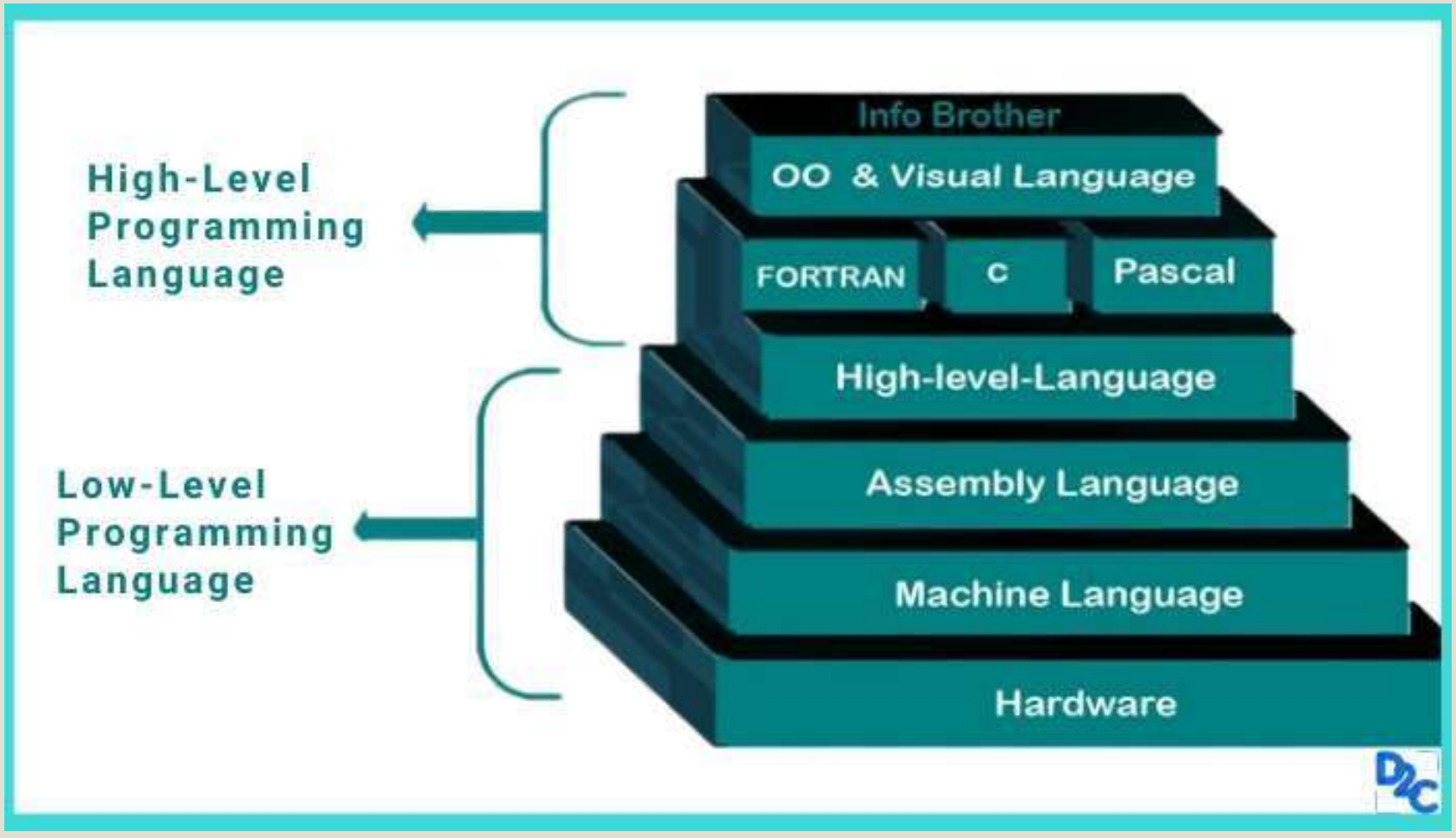
Q. During the execution of the instructions, a copy of the instructions is placed in the _____

a) Register
b) RAM
c) System heap
d) Cache

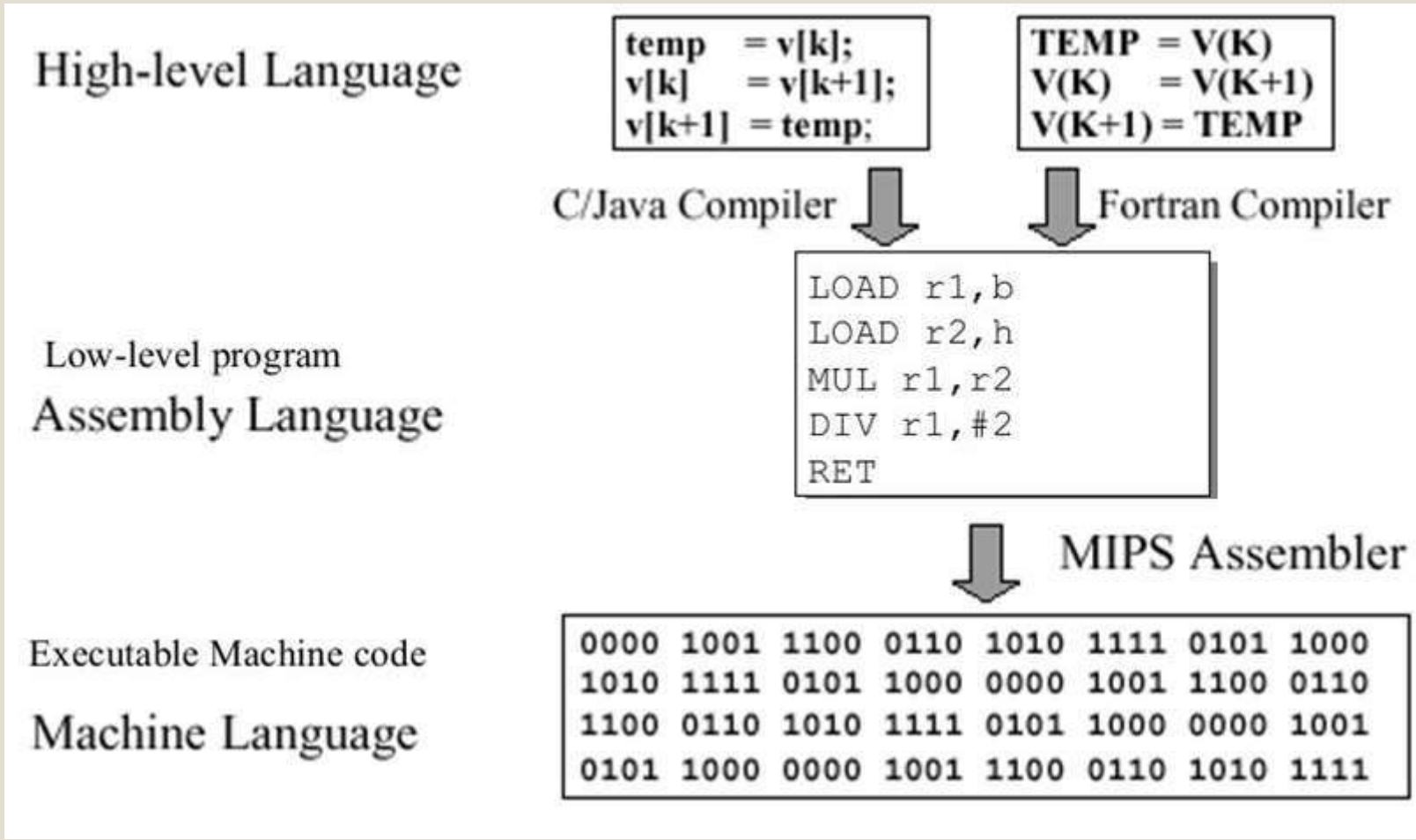
Programming languages



Translation from a High Level Language to the Hardware Language Instruction Set Architecture



CONT..


High-level Language
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
TEMP = V(K)
V(K) = V(K+1)
V(K+1) = TEMP
C/Java Compiler
Fortran Compiler
Low-level program
Assembly Language
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
MIPS Assembler
Executable Machine code
Machine Language
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

Levels of Programming Languages

High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

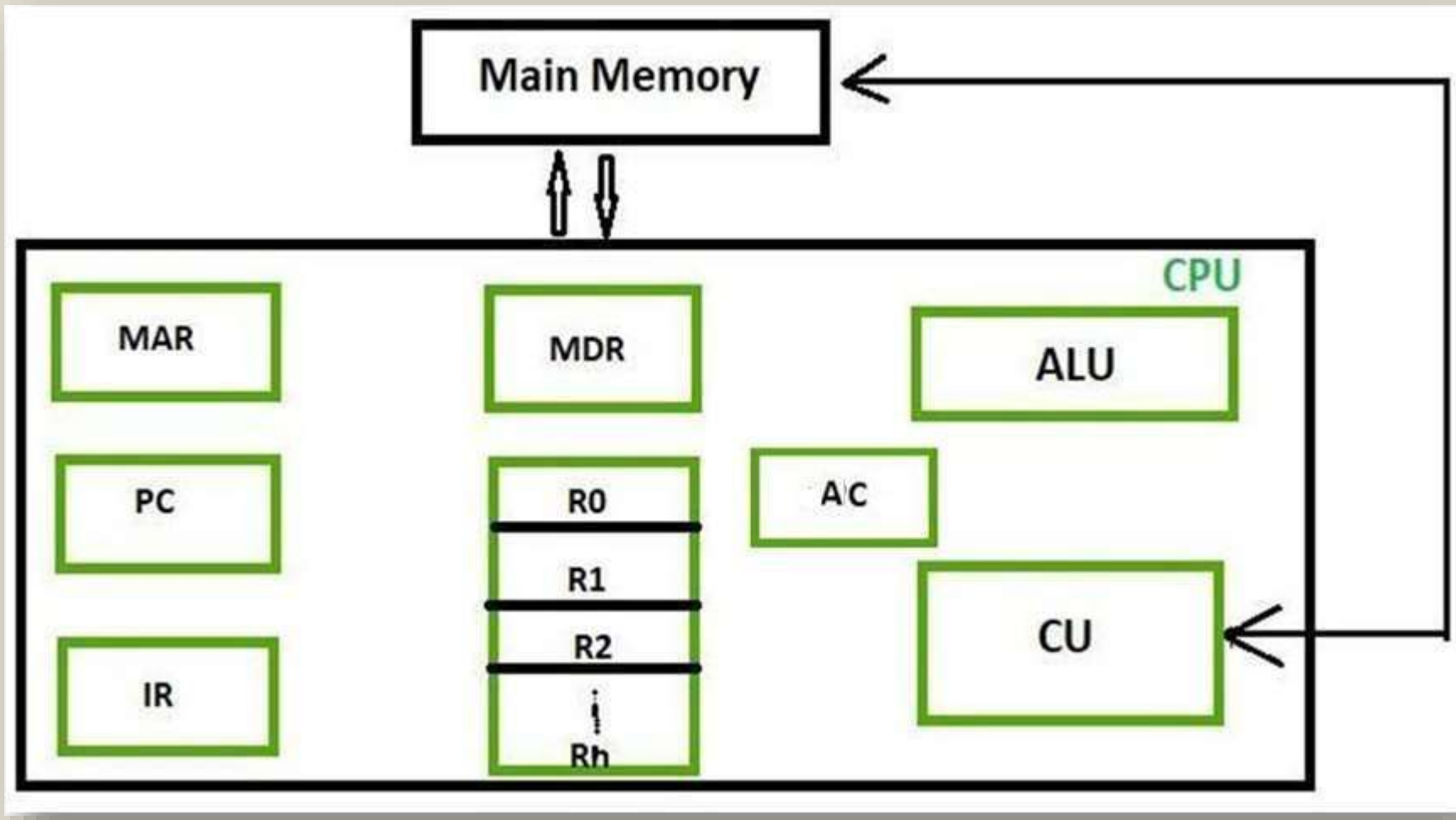
Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
0001001001000101
0010010011101100
10101101001...
```

Basic CPU structure



CPU Registers

- Registers are also called **internal memory or immediate access memory stores**. A register is a small amount of fast temporary memory within the processor where the ALU or the CU can store and change values needed to execute instructions. It is made up of flip-flops.

Types of CPU Registers and their Functions

Memory Address Register (MAR): It stores the memory locations of instructions that need to be fetched from memory or stored into memory.

- MAR helps to make the communication with using of MDR (Memory Data Register) in between the **CPU** and **Main Memory**.

Memory Data Register (MDR): It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.

- **CPU** fetches some mandatory instructions and data from **main memory** side then its temporary copy is saved into this data register before decoding this data. So, MDR register works as a middle buffer

CPU Registers

Accumulator: Main function of Accumulator Register is to store the output which is generated by your system. When CPU will execute some instruction then it will produce the result, now AC register is needed to store those produced data.

Program Counter (PC): Program Counter register's function is to hold all records in sequence of entire execution of programs. PC has the memory address of further instruction that is fetched in next step. PC registers to keep track the address of next instruction which to be fetched from the **primary memory**, if recently instruction is completely executed. It helps to count all numbers of entire instructions.

Current Instruction Register (CIR): It stores the most recently fetched instructions while it is waiting to be decoded and executed.

Instruction Buffer Register (IBR): The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

Cont..

GPR Registers

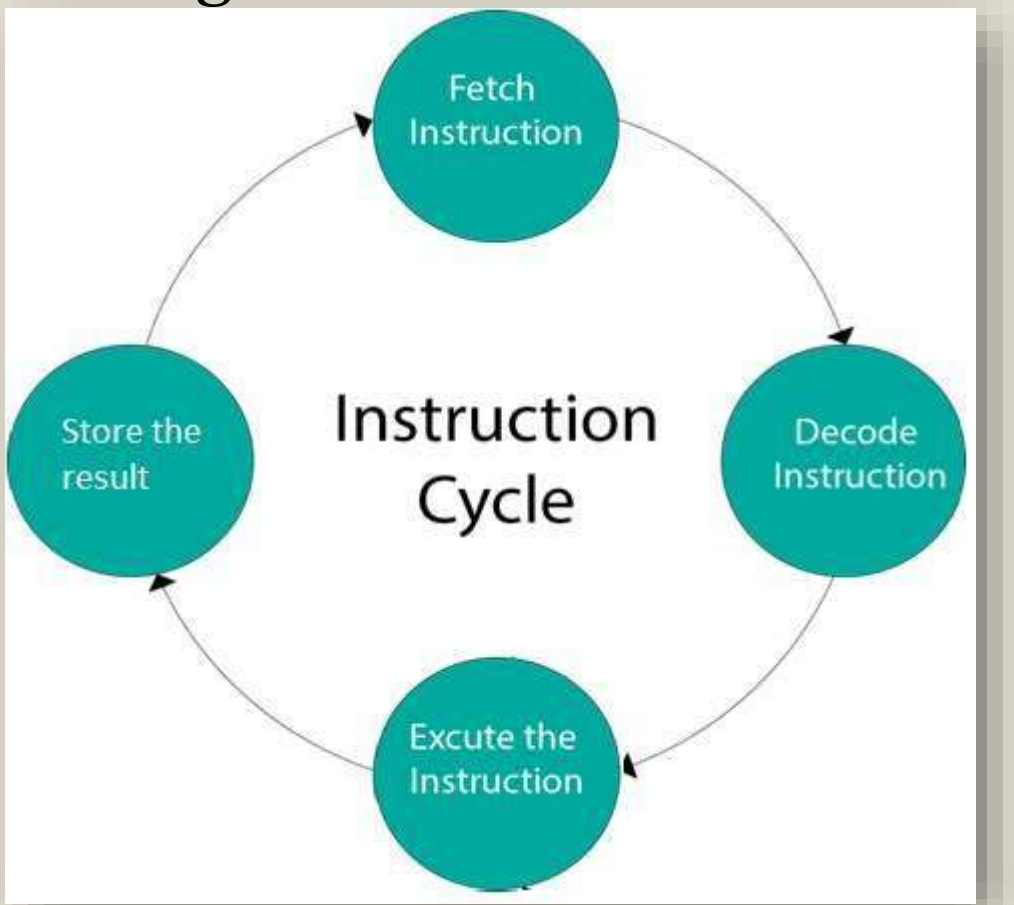
- GPR stands for "**General Purpose Registers**", and these are unified types of registers (R0, R1, R2 . .). These registers are capable to store the memory addresses, data values as well as floating-point values. Mostly, GPR registers are used into modern **CPU** and GPUs due to their best flexibility.
- The CPU has **8 general-purpose registers**, each capable of storing 32-digit binary numbers

Register and memory configuration

Register	Symbol	Number of bits	Function
Data register	DR	16	Holds memory operand
Address register	AR	12	Holds address for the memory
Accumulator	AC	16	Processor register
Instruction register	IR	16	Holds instruction code
Program counter	PC	12	Holds address of the instruction
Temporary register	TR	16	Holds temporary data
Input register	INPR	8	Carries input character
Output register	OUTR	8	Carries output character

Instruction cycle

- Software -> Programme -> Instructions



Basic Performance Equation

- **CPU Time = I * CPI * T**
  - I = number of instructions in program
  - CPI = average cycles per instruction
  - T = clock cycle time
- **CPU Time = I * CPI / F**
  - F = 1/T the clock rate (Frequency)

Q. Two processors A and B have clock frequencies of 700 Mhz and 900 Mhz respectively. Suppose A can execute an instruction with an average of 3 steps and B can execute with an average of 5 steps. For the execution of the same instruction which processor is faster?
a) A
b) B
c) Both take the same time
d) Insufficient information

Q. If a processor clock is rated as 1250 million cycles per second, then its clock period is _____

a) $1.9 * 10^{-10}$ sec
b) $1.6 * 10^{-9}$ sec
c) $1.25 * 10^{-10}$ sec
d) $8 * 10^{-10}$ sec

Q. The clock rate of the processor can be improved by _____

a) Improving the IC technology of the logic circuits
b) Reducing the amount of processing done in one step
c) By using the over clocking method
d) All of the mentioned

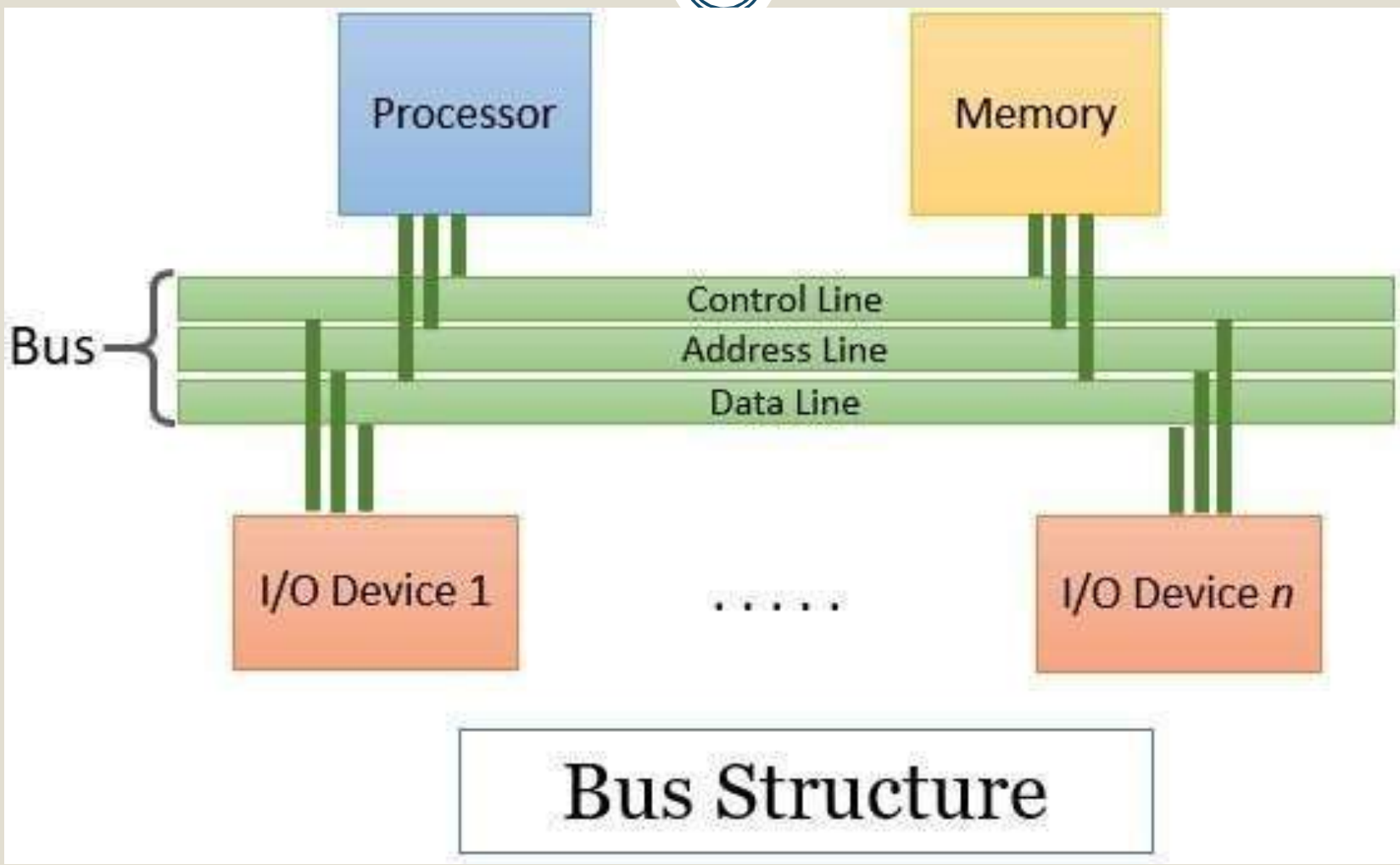
Bus Structure in Computer Architecture

- A group of wires that connected several devices is called a bus.
- Buses are used to send control signals and data between the processor and other components

Classification of bus:

i. Single bus system
ii. Multi bus system

Bus Structure



Bus Structure

1. Data Lines

- Data lines coordinate in transferring the data among the system components. The data lines are collectively called data bus.

- A data bus may have 32 lines, 64 lines, 128 lines, or even more lines. The number of lines present in the data bus defines the *width* of the data bus.

- Each data line is able to transfer only one bit at a time. So the number of data lines in a data bus determines how many bits it can transfer at a time. The performance of the system also depends on the width of the data bus.

- **All computers made in 2019** has **64 bits** data buses (64 bit machine)

2. Address Lines

- The content of the address lines of the bus determines the source or destination of the data present on the data bus.

  Address bus carry the memory address while reading from or writing into memory.

- The number of address lines together is referred to as address bus. The number of address lines in the address bus determines its *width*.

- The width of the address bus determines the memory capacity of the system.

- **E.g. : A system with a 32-bit address bus can address 2^{32} (4,294,967,296) memory locations**. If each memory location holds one byte, the addressable memory space is 4 GB.

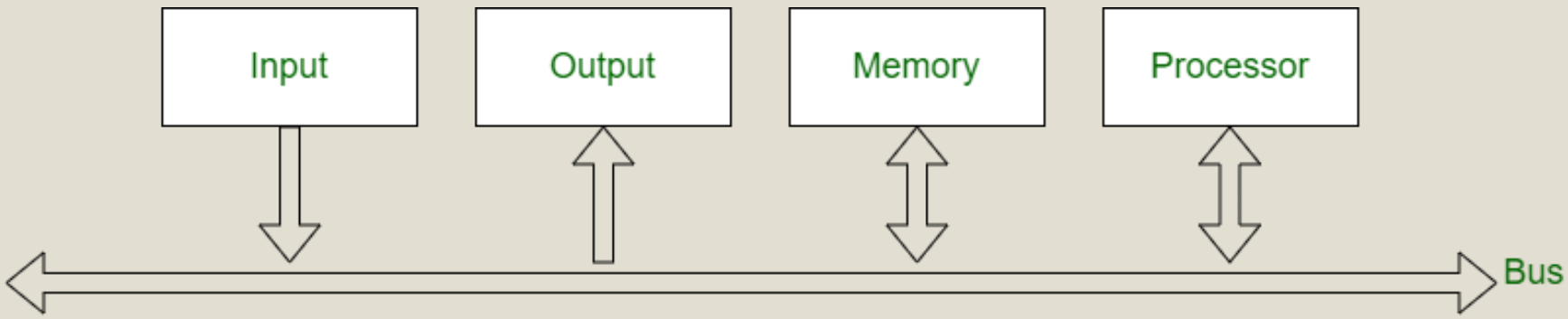
3. Control Bus

- The control signals placed on the control lines control the use and access to address and data lines of the bus
- The control signal consists of the *command* and *timing information*. Here the command in the control signal specify the *operation* that has to be performed. And the timing information over the control signals specify till when the data and address information is valid .

Different types of control signals

- ***Memory Read:*** This signal, is issued by the CPU or DMA (Direct access media) controller when performing a read operation with the memory.
- ***Memory Write:*** This signal is issued by the CPU or DMA controller when performing a write operation with the memory.
- ***I/O Read:*** This signal is issued by the CPU when it is reading from an input port.
- ***I/O Write:*** This signal is issued by the CPU when writing into an output port.
- ***Ready:*** The ready is an input signal to the CPU generated in order to synchronize the show memory or I/O ports with the fast CPU.

Single Bus Structure



Single Bus Structure

Single Bus

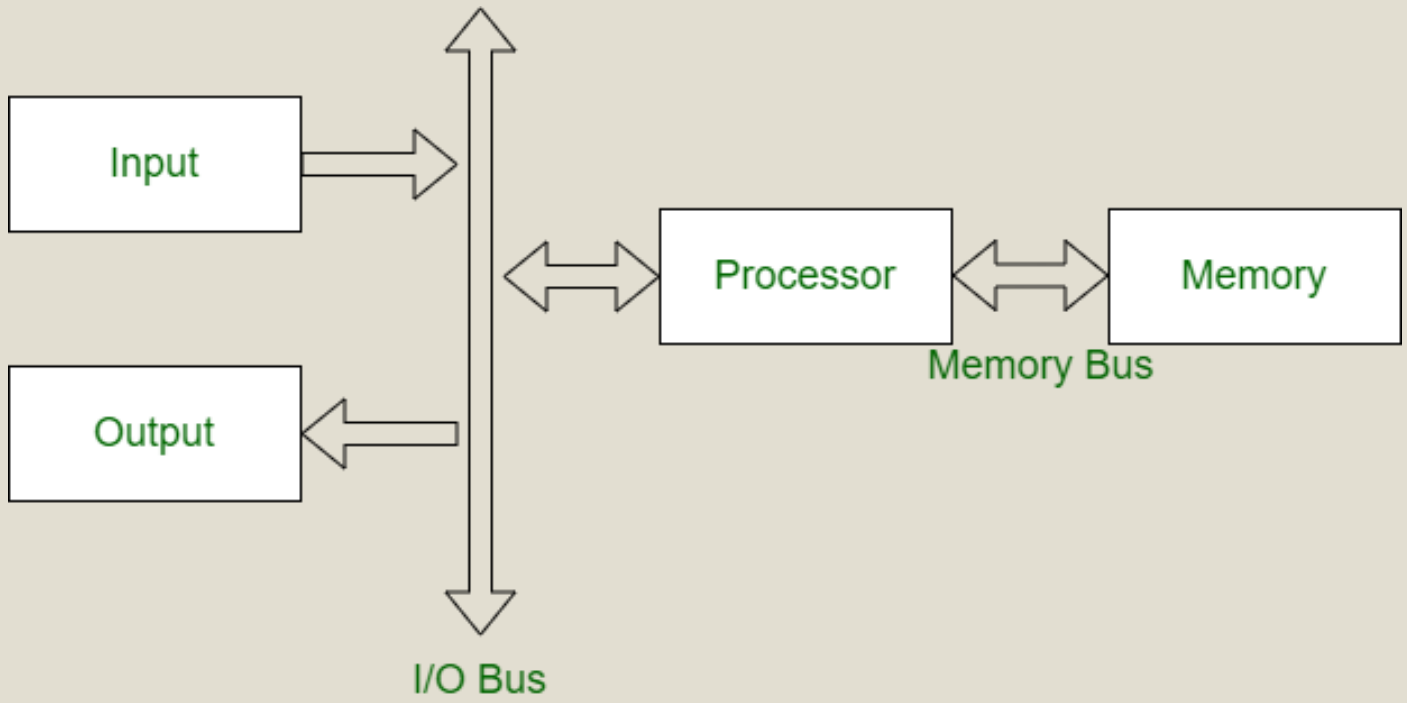
- **1. Single Bus Structure :**

  In single bus structure, one common bus used to communicate between peripherals and microprocessor. It has disadvantages due to use of one common bus.
- Four units share the single bus. At any given point of time, information can be transferred between any two units.
- Here I/O units use the same memory address space (Memory mapped I/O). So no special instructions are required to address the I/O, it can be accessed like a memory location
- Since all the devices do not operate at the same speed, it is necessary to smooth out the differences in timings among all the devices A common approach used is to include buffer registers with the devices to hold the information during transfers
- Ex: Communication between the processor and printer

Buffer Registers

- **Buffer Registers**
  → are included with the devices to hold the information during transfers.
  → prevent a high-speed processor from being locked to a slow I/O device during data transfers.

Double Bus Structure



Double Bus Structure

Double Bus

- **2. Double Bus Structure :**
  In double bus structure, one bus is used to fetch instruction while other is used to fetch data, required for execution. It is to overcome the bottleneck of single bus structure.

- I/O units are connected to the processor though an I/O bus and Memory is connected to the processor through the memory bus

- I/O bus consists of address, data and control bus, Memory bus also consists of address, data and control bus .In this type of arrangements processor completely supervises the transfer of information to and from I/O units. All the information is first taken to processor and from there to the memory . Such kind of transfers are called as program controlled transfer.

Difference between Single Bus Structure and Double Bus Structure

Single Bus Structure	Double Bus Structure
One common bus is used for communication between peripherals and processor.	Two buses are used, one for communication from peripherals and other for processor.
Instructions and data both are transferred in same bus.	Instructions and data both are transferred in different buses.
Its performance is low.	Its performance is high.
Cost of single bus structure is low.	Cost of double bus structure is high.
Number of cycles for execution is more.	Number of cycles for execution is less.
Execution of process is slow.	Execution of process is fast.
Number of registers associated are less.	Number of registers associated are more.
At a time single operand can be read from bus.	At a time two operands can be read.

Basic Computer Instructions

The set of instructions incorporated in 16- bit IR register are:

- Data transfer instructions
  - To move information to and from memory
- Data manipulation instructions
  - Arithmetic, logical and shift instructions
- Programme control instructions
  - Program control instructions with status conditions

Data transfer instructions

- **To transfer information to and from memory / Registers**

Name	Mnemonic Symbols
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	In
Output	OUT
Push	PUSH
Pop	POP

Data manipulation instructions

- It contains Arithmetic, logical and shift instructions

Name	Mnemonics
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Programme control instructions

- Program control instructions with status conditions

Name	Mnemonics
Branch	BR
Jump	JMP
Skip	SKP
Call	Call
Return	RET
Compare (by Subtraction)	CMP
Test (by ANDing)	TST

Program control instructions with status conditions

Symbol	Description
SPA	Skip next instruction if AC > 0
SNA	Skip next instruction if AC < 0
SZA	Skip next instruction if AC = 0
SZE	Skip next instruction if E = 0

Instruction Formats

A computer performs a task based on the instruction provided.

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

Based on the number of address, instructions are classified as:
Zero , one, two, three Address Instructions

Zero Address Instructions

A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

Infix	Reverse Polish notation
A + B × C	A B C × +
A × B + C	A B × C +
A × B + C × D	A B × C D × +
(A + B) / (C – D)	A B + C D – /
A × B / C	A B × C /
((A + B) × C + D)/(E + F + G)	A B + C × D + E F + G + /

Note : let's use X = (A+B)*(C+D) expression to showcase the procedure.

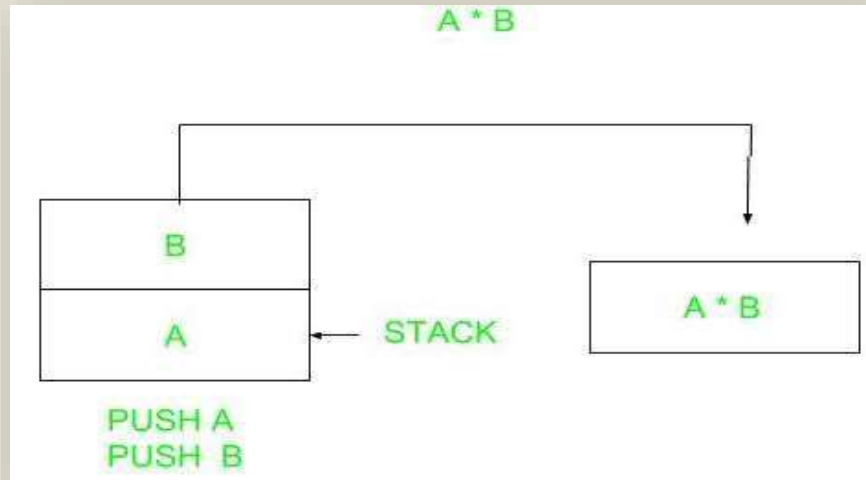
Opcode	Operand	Description
PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

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

Post fixed : X =

AB+CD+* TOP means top

of stack

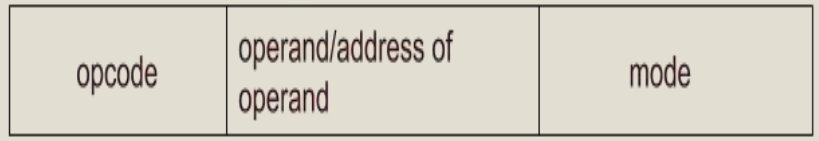


One Address Instructions

- This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location.
- Implied means that the CPU already knows that one operand is in the 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

Opcode	Operand	Description
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**



Two Address Instructions

- Here two addresses can be specified in the instruction. here the result can be stored at different locations rather than just in accumulators,

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

Opcode	Destination, source address	Description
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

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 instruction contains more information.
- but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

Opcode	Operand	Description
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**

CPU organization

Instruction is of variable length depending upon the number of addresses it contains

CPU organization is of three types based on the number of address fields:

- Stack organization.
- Single Accumulator organization.
- General register organization.

Stack-based CPU Organization

- The computers which use Stack-based CPU Organization are based on a data structure called a **stack**.
- The stack is a list of data words. It uses the **Last In First Out (LIFO)** access method.
- In this organization, ALU operations are performed on stack data. Both the operands are always required on the stack. After manipulation, the result is placed in the stack.
- Stack-based CPU organization uses zero address instruction.
- The main two operations that are performed on the operators of the stack are **Push** and **Pop**.

Single Accumulator organization

- The accumulator register is used implicitly for processing all instructions of a program and storing the results into the accumulator.
- The instruction format that is used by this CPU Organisation is the **One address field**.
- 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.
- Accumulator is the default address thus after data manipulation the results are stored into the accumulator.
- Mainly two types of operation are performed in a single accumulator based CPU organization: **Data transfer operation & ALU operation**
- For ex: LOAD X , AC <-- X
  MUL X AC <-- AC * M[X]

- **1. 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 a memory read operation that is data is transferred from memory to accumulator and STORE is a memory write operation that is data is transferred from the accumulator to memory.
- **2. 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

Advantages –

- One of the operands is always held by the accumulator register. This results in short instructions and less memory space.
- The instruction cycle takes less time because it saves time in instruction fetching from memory.

Disadvantages –

- When complex expressions are computed, program size increases due to the usage of many short instructions to execute it. Thus memory size increases.
- As the number of instructions increases for a program, the execution time increases.

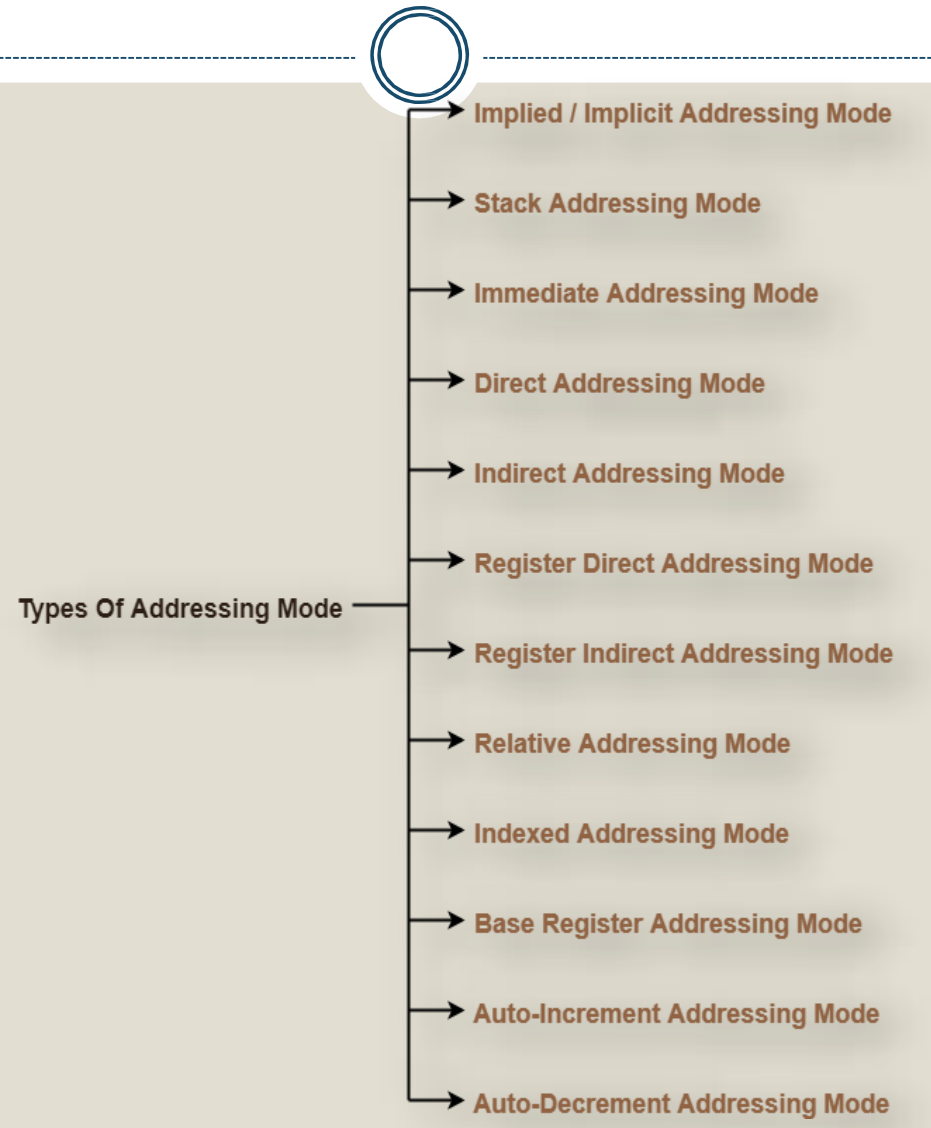
General register organization.

- When we are using multiple general-purpose registers, instead of a single accumulator register, in the CPU Organization then this type of organization is known as General register-based CPU Organization.
- In this type of organization, the computer uses two or three address fields in their instruction format. Each address field may specify a general register or a memory word.
- E.g. MULT R1, R2, R3 R1 <-- R2 * R3
- MULT R1, R2 R1 <-- R1 * R2

Addressing Modes

- An **addressing mode** is the process used by a microprocessor to deliver instructions to a machine so operations can be performed.
- The **addressing mode** is the method to specify the operand of an instruction.

Types of addressing modes



Implicit addressing mode

- In Implicit addressing mode no operand (register or memory location or data) is specified in the instruction. As in this mode the operand are specified implicit in the definition of instruction.
- all Register-Reference instruction that use an accumulator and Zero-Address instruction in a Stack Organised Computer are implied mode instructions.

- Eg. **CMA**: Take complement of content of AC

  RLC: Rotate the content of Accumulator is an implied mode instruction.

Stack addressing mode

- In this mode, operand is at the top of the stack.
- E.g : ADD , MUL

This instruction will add the top two items from the stack and will then *PUSH* the result to the top of the stack.

Immediate addressing 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**.

- Example: MOV AL, 35H (move the data 35H into AL register)

- **MVI 06** Move 06 to the accumulator
- **ADD 05** ADD 05 to the content of accumulator

- Here 06, 05 are the operands not the addresses.

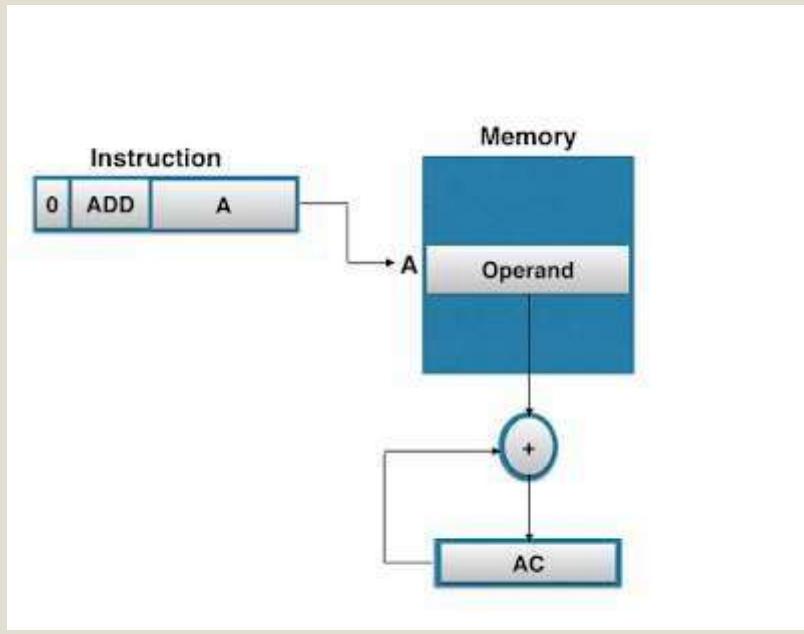
Direct addressing mode

- In this mode the address of data (operand) is specified in the instruction itself.
- Single memory reference to access data.
- In this type of mode, the operand resides in memory and its address is given directly by the address field of the instruction.

e.g.

ADD A :Means add contents of cell A to accumulator .

ADD R1, 4000 - In this the 4000 is effective address of operand.



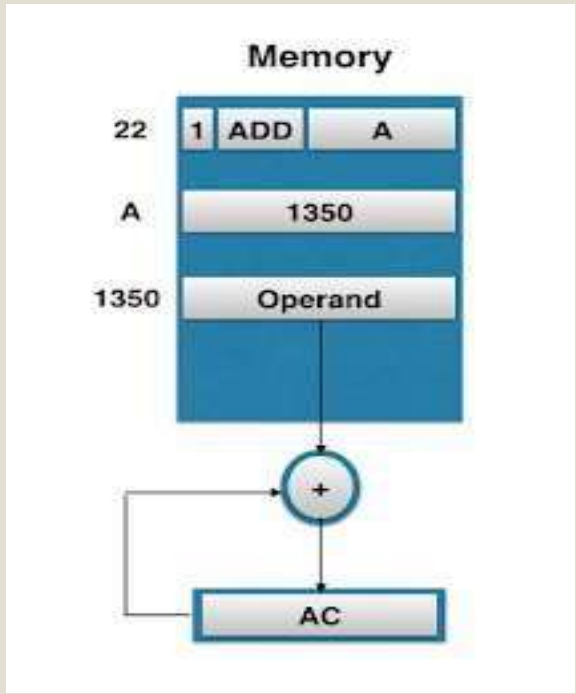
Indirect addressing mode

- 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.

E.g ADD (A)
Means adds the content of cell pointed to contents of A to Accumulator

Thus in it, AC <-- M[M[A]] [M=Memory]
i.e., (A)=1350=EA

LOAD R1, (1005)
or LOAD R1, @1005



Register Direct addressing mode

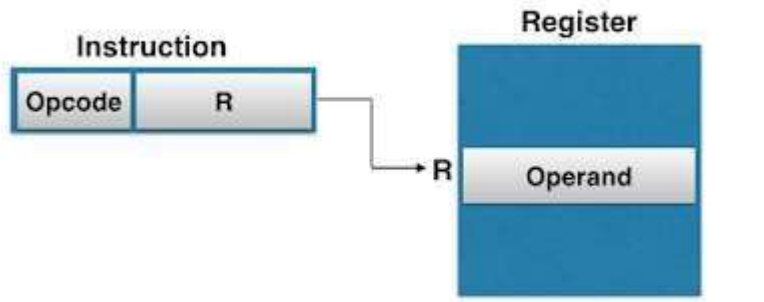
- In Register Addressing Mode, the operands are in registers that reside within the CPU. That is, in this mode, instruction specifies a register in CPU, which contain the operand.
- It is like Direct Addressing Mode, the only difference is that the address field refers to a register instead of memory location.
- operand is placed in one of 8 bit or 16 bit general purpose registers

e.g. 7

MOV AX, BX Move contents of Register BX to AX

ADD AX, BX Add the contents of register BX to AX

Here, AX, BX are used as register names which is of 16-bit register.

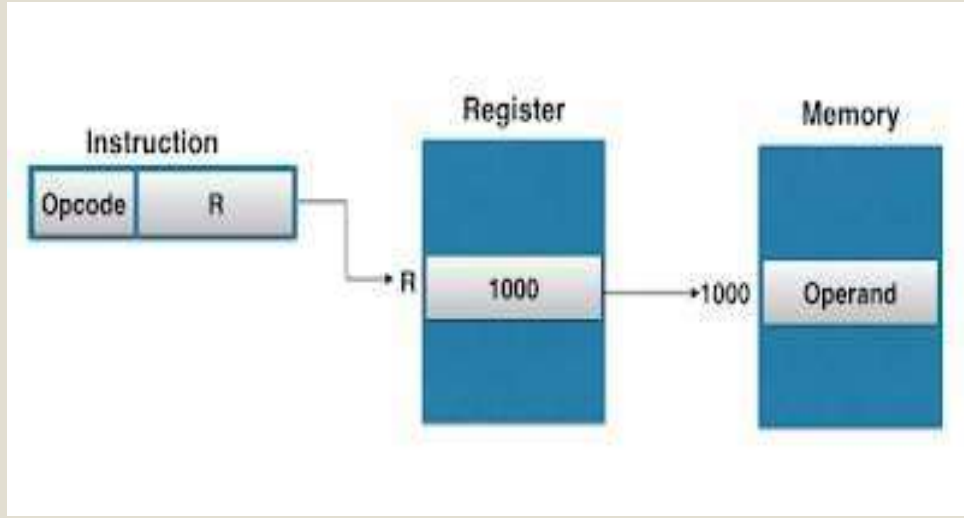


Register indirect addressing mode

- The instruction specifies a register in CPU whose contents give the operand in memory. In other words, the selected register contain the **address of operand** rather than the operand itself

MOV AX, [BX]

(move the contents of memory location addressed by the register BX to the register AX)

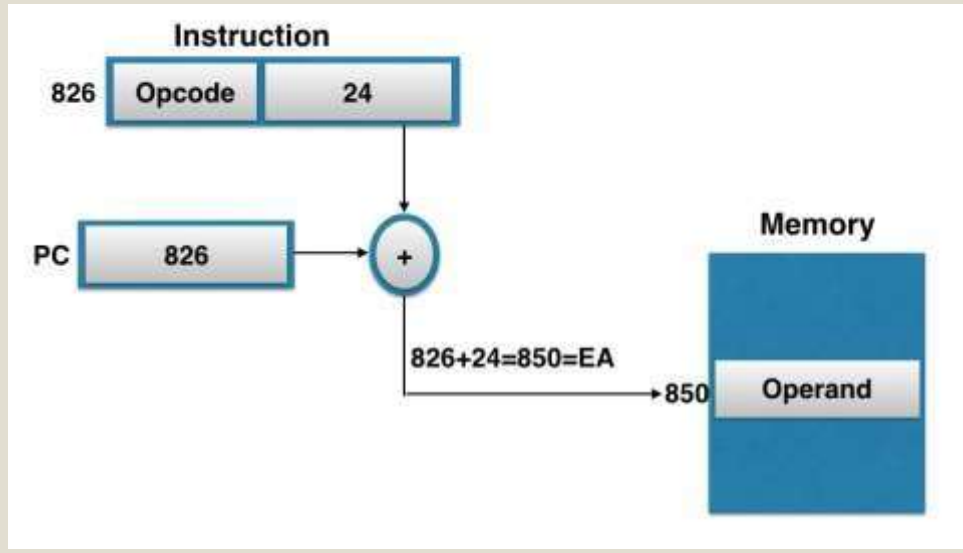


Code example in Register:

MOV BX, 1000H

Relative addressing mode

- In Relative Addressing Mode , the contents of program counter is added to the address part of instruction to obtain the Effective Address.
- i.e. EA=A+PC



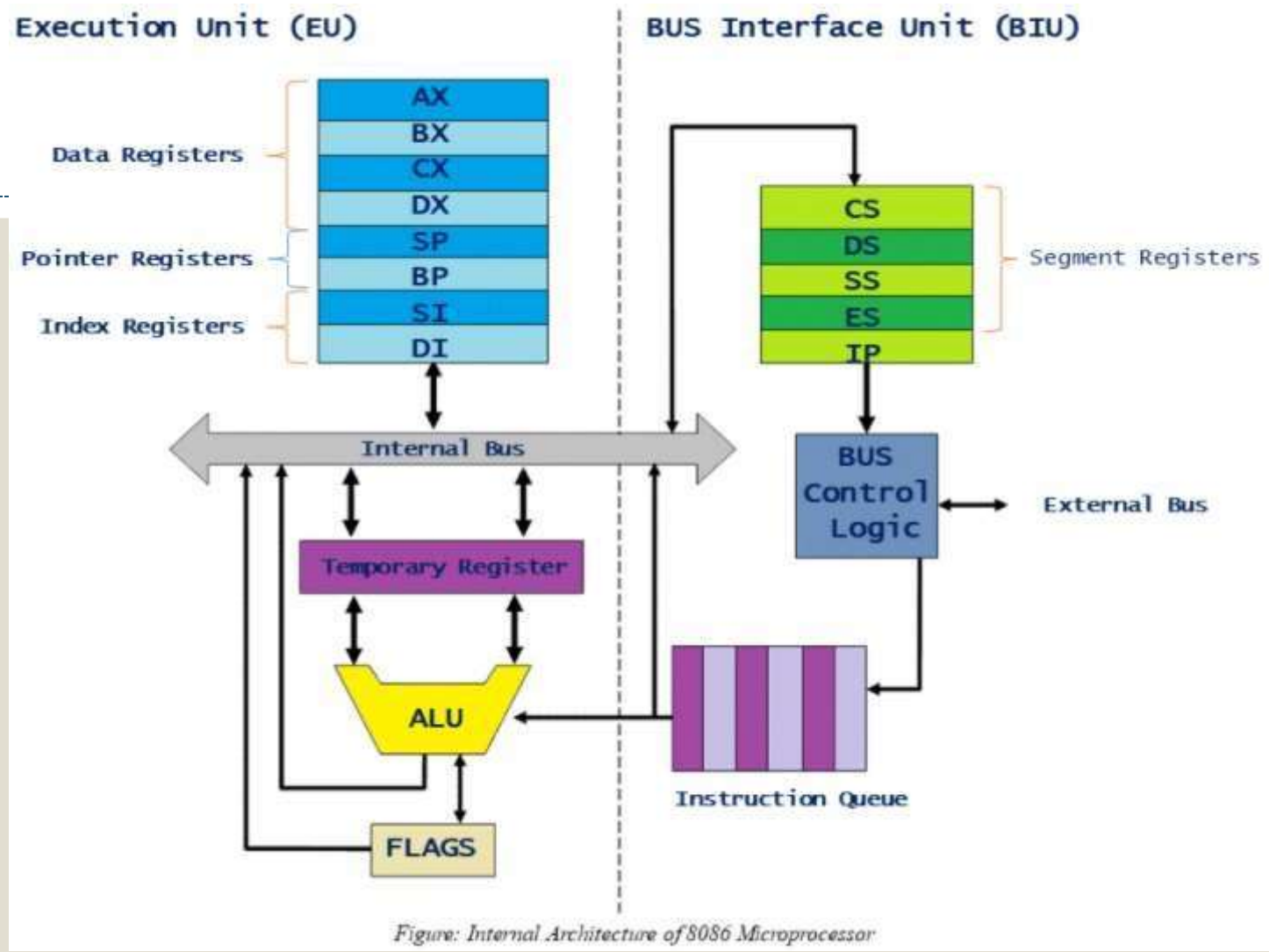


Figure: Internal Architecture of 8086 Microprocessor

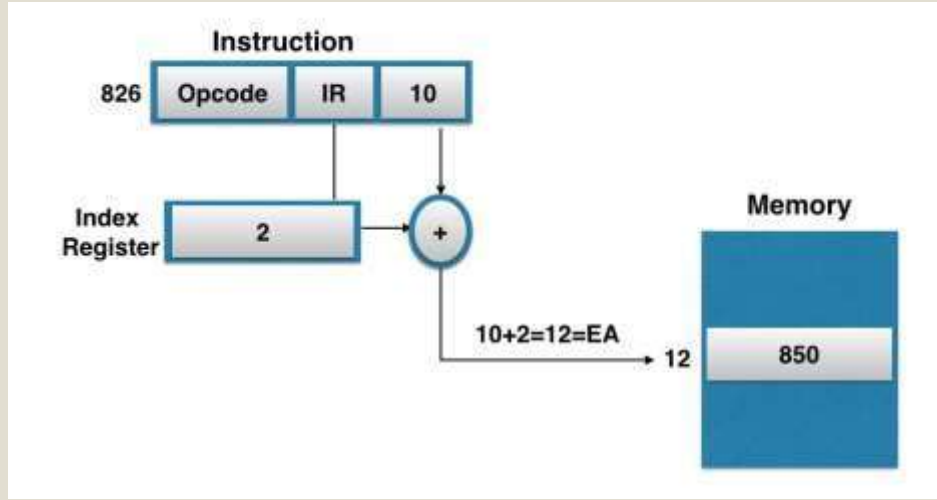
- An index register holds **the current offset of a memory location**, with another register holding the base address, so the combination of the two registers creates a completed memory address.

Indexed Register addressing mode

- The content of Index Register is added to direct address part (or field) of instruction to obtain the effective address.
- Indexed register is a special CPU register that contains the **offset address** where as direct addressing field contain base address.
- Used to access or implement the array efficiently.

- EA= Index Register + Base address

```
MOV AX, [SI + 05]
```

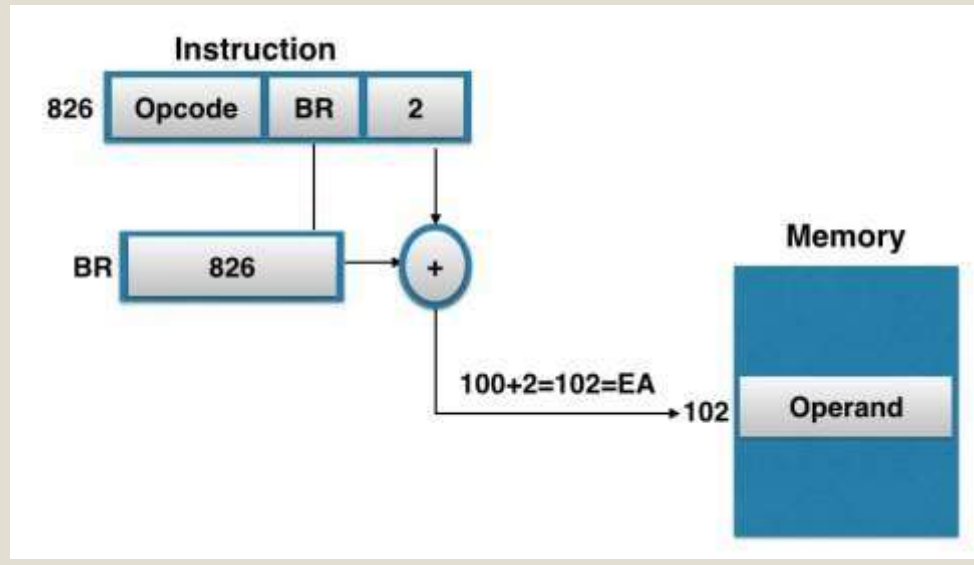


Base register addressing mode

- The content of the Base Register is added to the direct address part of the instruction to obtain the effective address.
- Means, in it the register indirect address field point to the Base Register and to obtain EA, the contents of Instruction Register, is added to direct address part of the instruction
- Used in programme relocation .

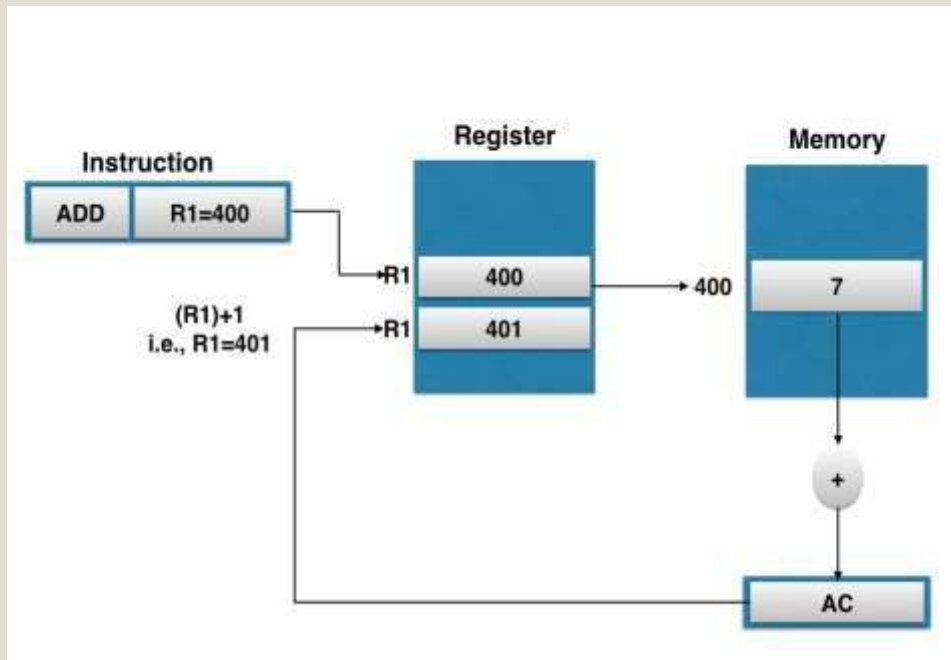
- EA=Base register + Address field value

```
ADD AX, [BX+SI]
```



Auto increment addressing mode

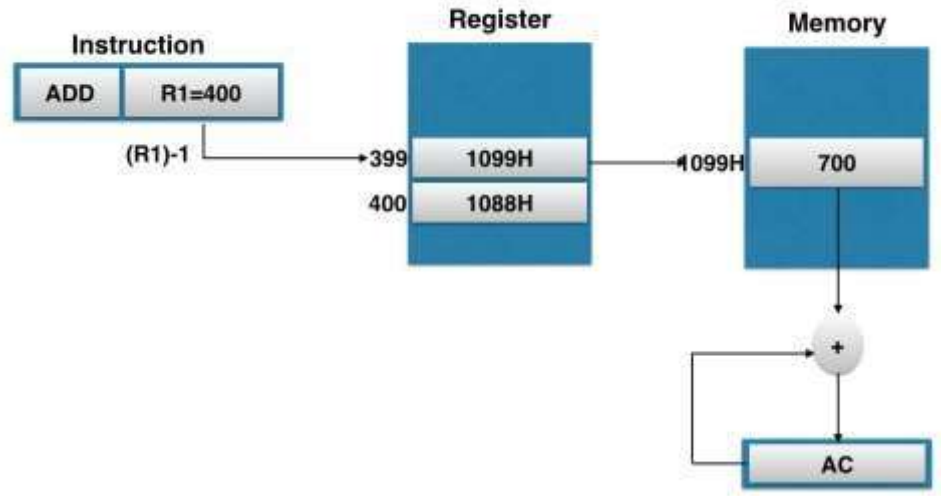
- Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.**(R1)+**.
- E.g.
- ->Add (R1)+ :
  - AC= AC+ R1,
  - R1= R1 + d
- ->Add R1, (R2)+ // OR
  - R1 = R1 +M[R2]
  - R2 = R2 + d
- *d* – size of an element



Auto decrement addressing 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)**

- **E.g** Add R1,-(R2) //OR
- R2 = R2-*d*
- R1 = R1 + M[R2]



QSN. In the following indexed addressing mode instruction, MOV 5(R1), LOC the effective address is _____

a) EA = 5+R1
b) EA = R1
c) EA = [R1]
d) EA = 5+[R1]

Register transfer notation / Language

- Each instruction consists of a sequence of smaller instructions called micro operations.
- Exact sequence of the micro operations that are carried out by an instruction can be specified using register transfer language (RTL)/ notation (RTN).
- It is not executed by the computer. It is used to explain how the computer works.

Eg. ADD R1, R2 : RTN: R1 <- R1 + R2

Qsn: Represent the following instructions in RTN.

- Add R1, R2, R3
- Store B

Examples

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

RISC Vs CISC

- **Reduced Instruction Set Architecture (RISC) –** The main idea behind this is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, a store command will store the data.

- **Complex Instruction Set Architecture (CISC) –** The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it's complex.

- Both approaches try to increase the CPU performance

RISC Vs CISC

RISC (Reduced instruction set computer)	CISC (complex instruction set computer)
Small number of fixed length instructions	Large number of Variable sized instructions
It supports Few no of addressing modes	It supports Large no of addressing modes
Cost is less because hardware supports less no of instructions.	Cost is more because hardware supports more instructions.
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM operations
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small
An instruction executed in a single clock cycle	Instruction takes more than one clock cycle
An instruction fit in one word	Instructions are larger than the size of one word
Hardware control unit	Microprogramme based control unit

- E.g.

- RISC: Apple- iPod, Smartphone, tablets.
- CISC: Intel X86 , Motorola 68000

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

CISC is reducing

RISC is reducing

Speed up

- Speed up S(n) =

Time taken to execute the programme using single processor

Time taken to execute the programme using "n" no of processors.

e.g. : T(1) = 1sec & if n= 2 T(2) = ½ sec = 0.5 sec

Hence S (n) = T(1) / T(2) = 1/ 0.5 = 2

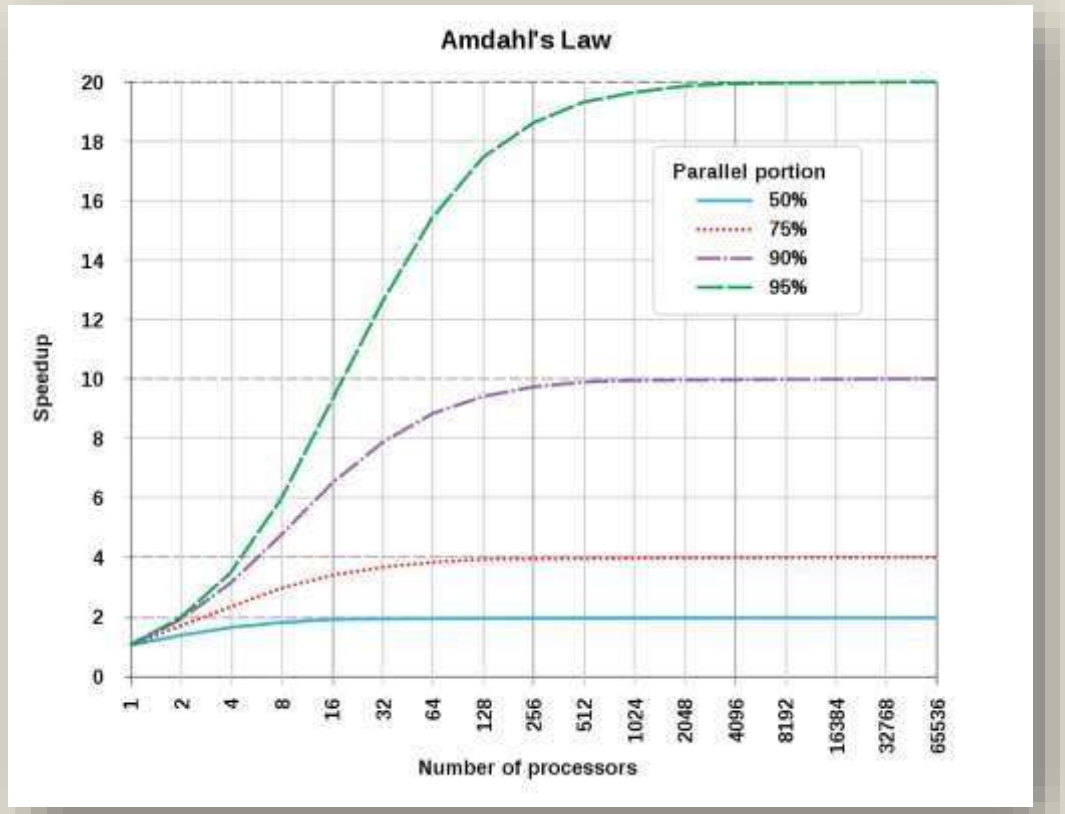
Means the speed up increases by 2 times

Amdahl's law

- Amdahl's law is an expression used to find the maximum expected improvement to an overall system when only part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speedup using multiple processors.
- Speedup metric is a quantitative measure of performance , which defines the benefits of running a programme in parallel.
- It states that a small portion of the programme which can't be parallelized (serial part) , will limit the overall speedup , available from penalization.
- typically any large mathematical engineering problem consists of several parallizable parts and several serial parts
- the speedup of a programme using multiple processors in parallel computing is limited by the time needed for the sequential fraction(non-parallelizable) part of the programme.

Cont.

Amdahl's law tells that for a given problem size , the speedup doesn't increase linearly as the no of processors are increasing. In fact the speedup tends to become saturated.



- Let the fraction of computation that can be executed in serial is ‘f’ (serial section) & the remaining computation that can be executed in parallel is (1-f).

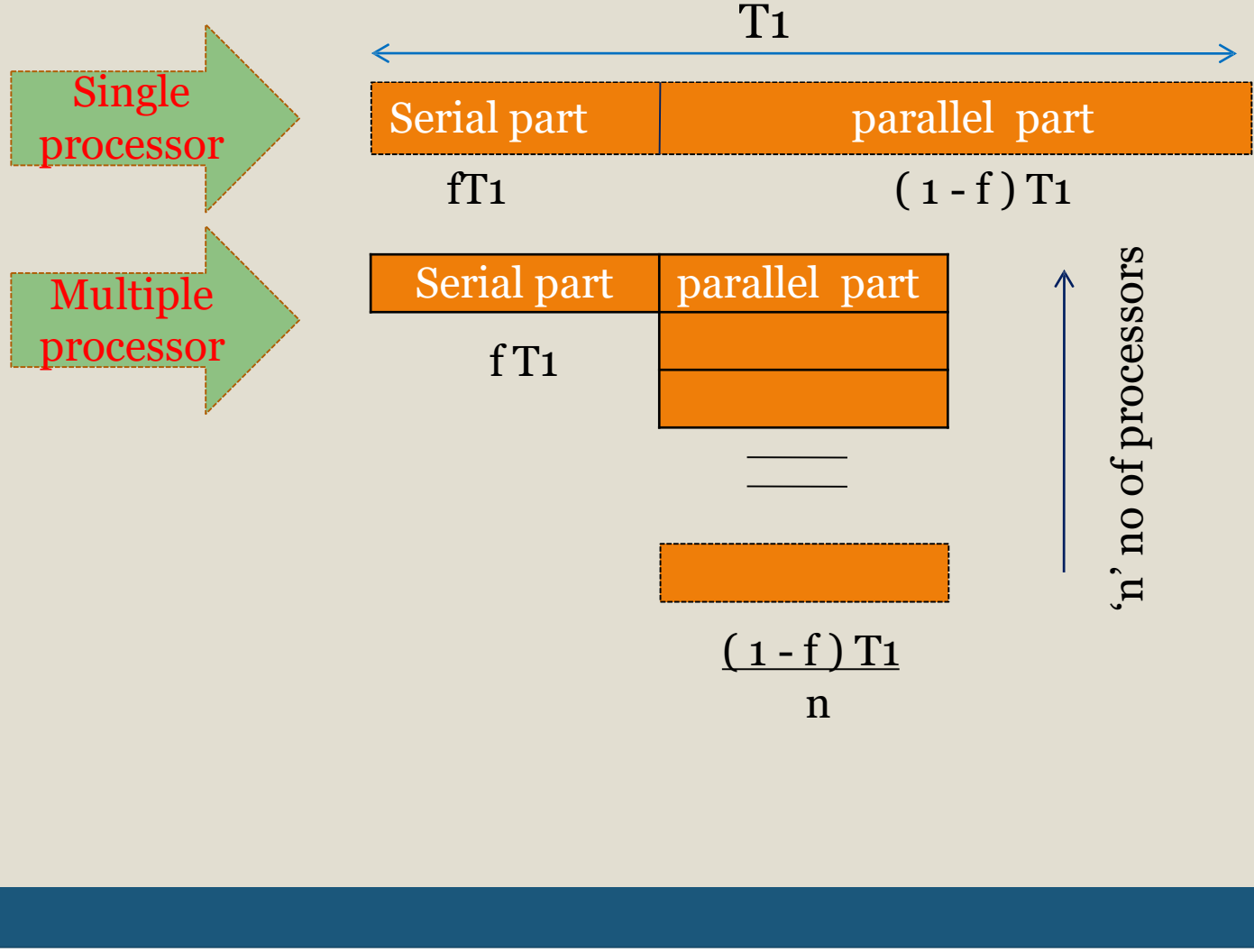
- Then the time to perform the computation with single processor (T1) & multiple processor (Tn) is given by

Time taken by the processor = Time required for serial part + Time required for serial part

For one processor $T_1 = f\,T_1 + (1 - f)\,T_1$

For ‘n’ no of processor $T_n = f\,T_1 + \frac{(1 - f)\,T_1}{n}$

Cont.


T1
Single processor
Serial part
parallel part
fT1
(1 - f) T1
Multiple processor
Serial part
parallel part
f T1
'n' no of processors
(1 - f) T1
n

Cont.

- Speed up S(n) =

  Time taken to execute the programme using single processor

  Time taken to execute the programme using “n” no of processors.

- S(n) = T1/Tn

$$= \frac{f\,T1 + (1 - f)\,T1}{f\,T1 + \frac{(1 - f)\,T1}{n}}$$

- S(n) = 1 / (f + ($\frac{1 - f}{n}$)) -> Formula for speedup

Questions(2 Marks)

1. Define Computer architecture and computer organization.
2. Give the different functional units of digital computer.
3. Define Amdhal's law.
4. Write the addressing mode and instruction format used the instruction ADD (R3), R1
5. Define Bus? What is the use of buffer register?
6. Compare single bus structure and multiple bus structure?
7. What is System Software? Give an example?
8. What is processor execution time of a program?
9. Write down the basic performance equation?
10. List out the various registers are used in basic operational concept?

Questions (5 Marks)

1. Draw and explain a Von-Neumann architecture.
2. Write the functions of MAR, MDR, IR and PC.
3. Define a machine instruction. How does it differ from user program?
4. Write down different types of Instruction set architecture.
5. What is the difference between RISC and CISC?
6. Calculate CPI of a computer which need 200 M hz frequency to operate if number of instruction of a program is 75 and each require 3 number of average steps to execute.
7. Define Amdahl's law with suitable example.
8. Define addressing mode of an instruction with example.
9. What are different types of instruction format in machine instruction?
10. Write about auto-increment addressing mode with example.
11. Represent the following instructions in RTN.
    1. Add R1, R2, R3
    2. Store B
12. An instruction is stored at location 300 with its address field at location 301. The address field at location 301. The address field has the value 400. A processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is (i) Direct (ii) Immediate (iii) Relative (iv)Register indirect (v) Index with R1 as the index register

13. Write the assembly language notation and corresponding RTN for the following expression, if the system is using single accumulator processor and the system using stack operation.

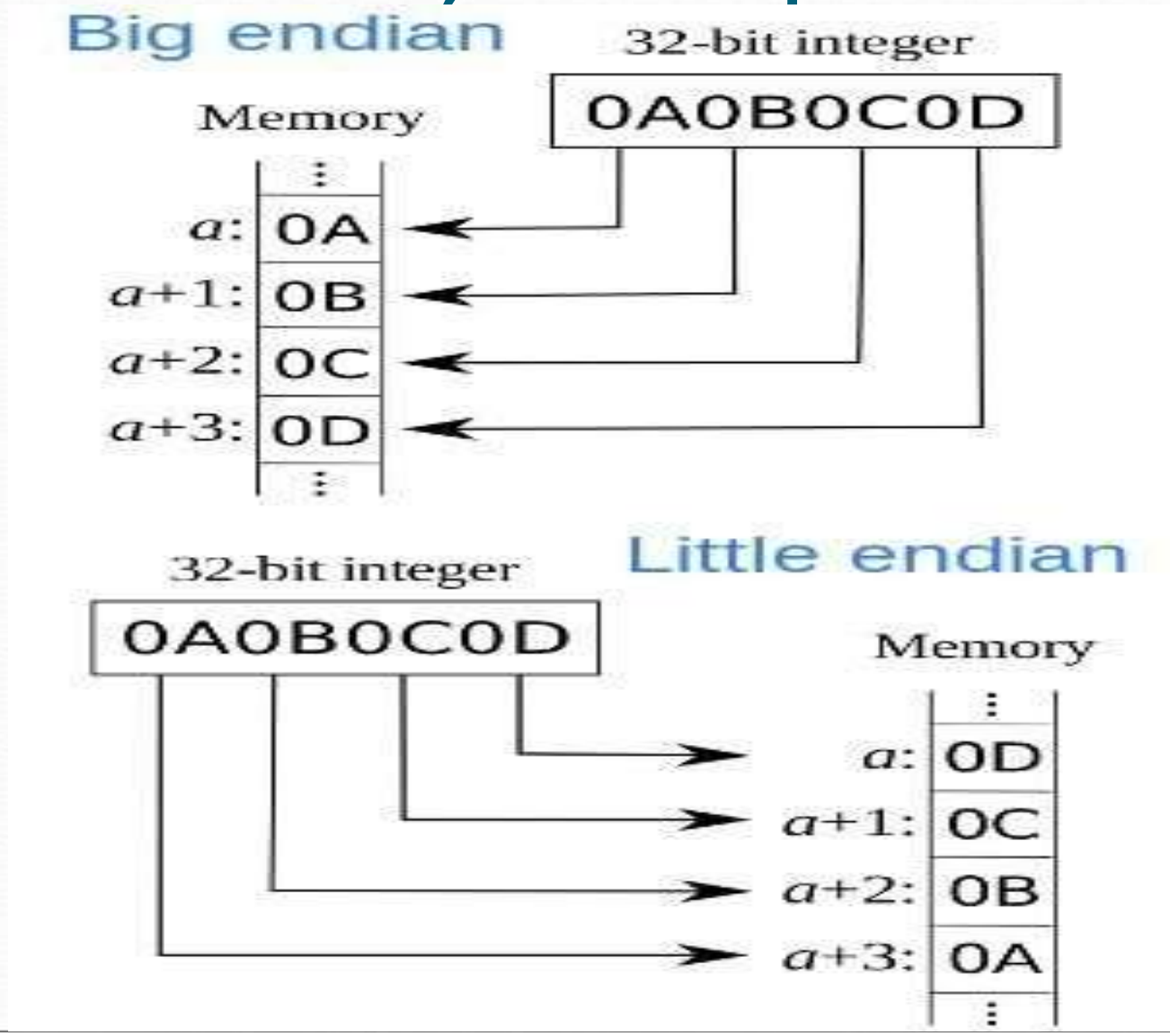
    Z = (A+B) X (C+B)
    E = (A X B) + (C X D)

Thank you

Memory Location, Addresses, and Operation

- Little and big endian are two ways of storing multibyte data-types (int, float, etc).
- In little endian machines, last byte of binary representation of the multibyte data-type is stored first.
- On the other hand, in big endian machines, first byte of binary representation of the multibyte data-type is stored first.

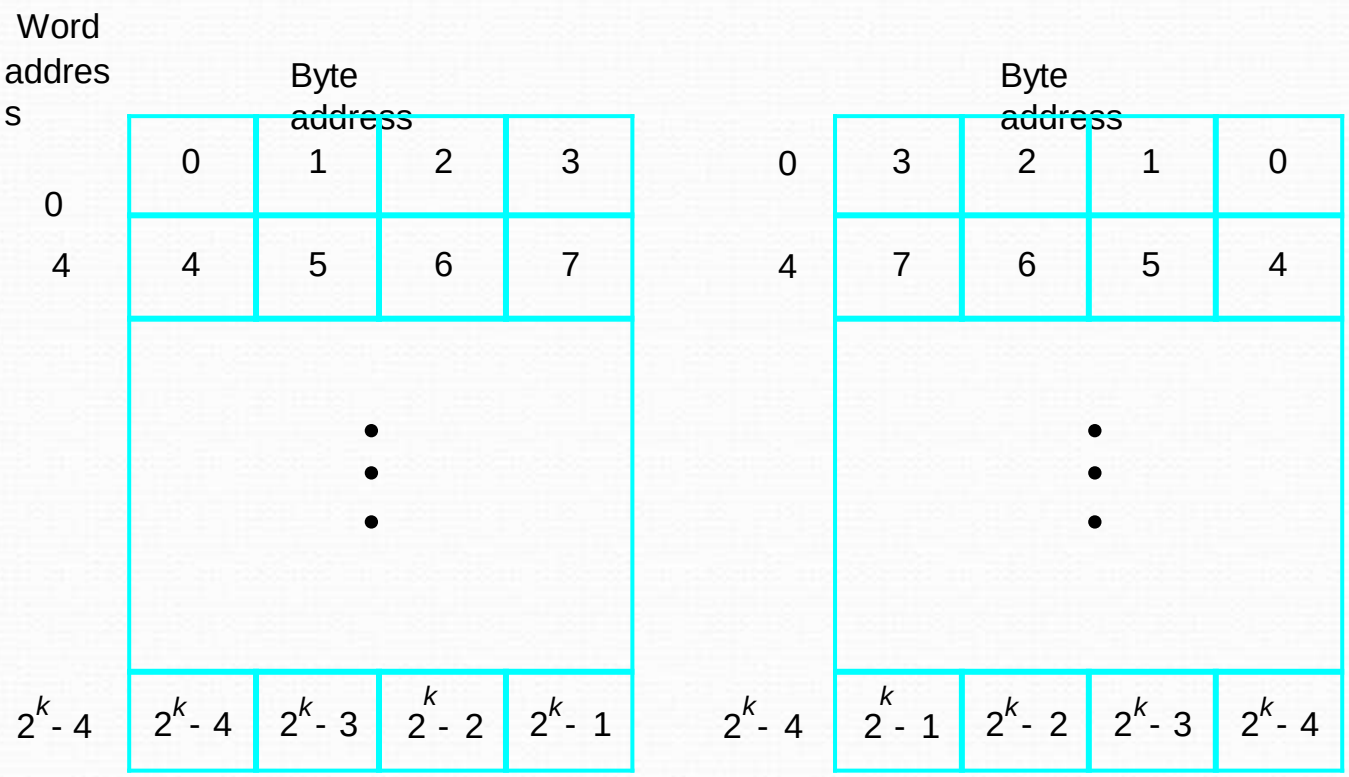
Memory Location, Addresses, and Operation



Big-Endian and Little-Endian Assignments

Big-Endian: lower byte addresses are used for the most significant bytes of the word

Little-Endian: opposite ordering. lower byte addresses are used for the less significant bytes of the word



(a) Big-endian assignment

(b) Little-endian assignment

Figure 2.7. Byte and word addressing.

Memory Location, Addresses, and Operation

- Address ordering of bytes
- Word alignment
  - Words are said to be aligned in memory if they begin at a byte addr. that is a multiple of the num of bytes in a word.
    - 16-bit word: word addresses: 0, 2, 4,....
    - 32-bit word: word addresses: 0, 4, 8,....
    - 64-bit word: word addresses: 0, 8,16,....
- Access numbers, characters, and character strings

Memory Operation

- Load (or Read or Fetch)
  - Copy the content. The memory content doesn't change.
  - Address – Load
  - Registers can be used
- Store (or Write)
  - Overwrite the content in memory
  - Address and Data – Store
  - Registers can be used

Chapter-2:Instruction Set Architecture

Machine Instruction

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Register Transfer Notation

- Identify a location by a symbolic name standing for its hardware binary address (LOC, R0,…)
- Contents of a location are denoted by placing square brackets around the name of the location (R1←[LOC], R3 ←[R1]+[R2])
- Register Transfer Notation (RTN)

Assembly Language Notation

- Represent machine instructions and programs.
- Move LOC, R1 = R1←[LOC]
- Add R1, R2, R3 = R3 ←[R1]+[R2]

CPU Organization

- Single Accumulator
  - Result usually goes to the Accumulator
  - Accumulator has to be saved to memory quite often
- General Register
  - Registers hold operands thus reduce memory traffic
  - Register bookkeeping
- Stack
  - Operands and result are always in the stack

Instruction Formats

- Three-Address Instructions
  - ADD R1, R2, R3 R1 ← R2 + R3
- Two-Address Instructions
  - ADD R1, R2 R1 ← R1 + R2
- One-Address Instructions
  - ADD M AC ← AC + M[AR]
- Zero-Address Instructions
  - ADD TOS ← TOS + (TOS – 1)
- RISC Instructions
  - Lots of registers. Memory is restricted to Load & Store

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- Three-Address
  1. ADD R1, A, B ; R1 ← M[A] + M[B]
  2. ADD R2, C, D ; R2 ← M[C] + M[D]
  3. MUL X, R1, R2 ; M[X] ← R1 * R2

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- Two-Address

1. MOV R1, A ; R1 ← M[A]
2. ADD R1, B ; R1 ← R1 + M[B]
3. MOV R2, C ; R2 ← M[C]
4. ADD R2, D ; R2 ← R2 + M[D]
5. MUL R1, R2 ; R1 ← R1 * R2
6. MOV X, R1 ; M[X] ← R1

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- One-Address
  1. LOADA ; AC ← M[A]
  2. ADD B ; AC ← AC + M[B]
  3. STORE T ; M[T] ← AC
  4. LOADC ; AC ← M[C]
  5. ADD D ; AC ← AC + M[D]
  6. MUL T ; AC ← AC * M[T]
  7. STORE X ; M[X] ← AC

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- Zero-Address

1. PUSH A ; TOS ← A
2. PUSH B ; TOS ← B
3. ADD ; TOS ← (A + B)
4. PUSH C ; TOS ← C
5. PUSH D ; TOS ← D
6. ADD ; TOS ← (C + D)
7. MUL ; TOS ← (C+D)*(A+B)
8. POP X ; M[X] ← TOS

Instruction Formats

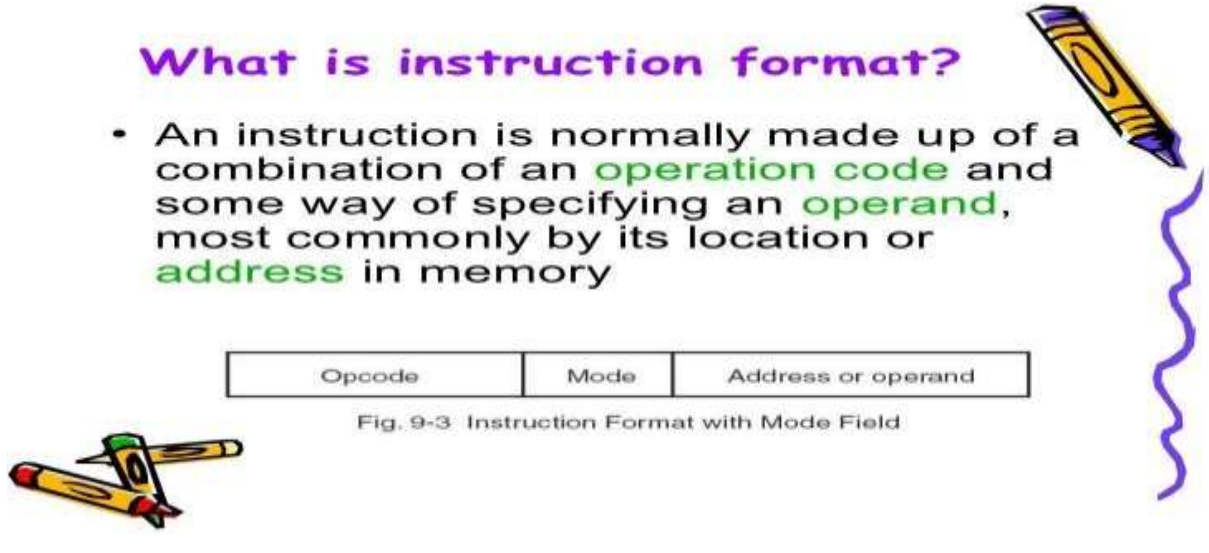
Example: Evaluate (A+B) * (C+D)

- RISC

1. LOADR1, A ; R1 ← M[A]
2. LOADR2, B ; R2 ← M[B]
3. LOADR3, C ; R3 ← M[C]
4. LOADR4, D ; R4 ← M[D]
5. ADD R1, R1, R2 ; R1 ← R1 + R2
6. ADD R3, R3, R4 ; R3 ← R3 + R4
7. MUL R1, R1, R3 ; R1 ← R1 * R3
8. STORE X, R1 ; M[X] ← R1

Accumulator: An accumulator is a register for short-term, intermediate storage of arithmetic and logic data in a computer's CPU (central processing unit). ... The most elementary use for an accumulator is adding a sequence of numbers.

Instruction format: An Instruction format must include an opcode, and address is dependent on an availability of particular operands. The format can be implicit or explicit which will indicate the addressing mode for each operand.



Opcode: Specifies the operation to be performed

Address Field: Designates a memory address or a processor register

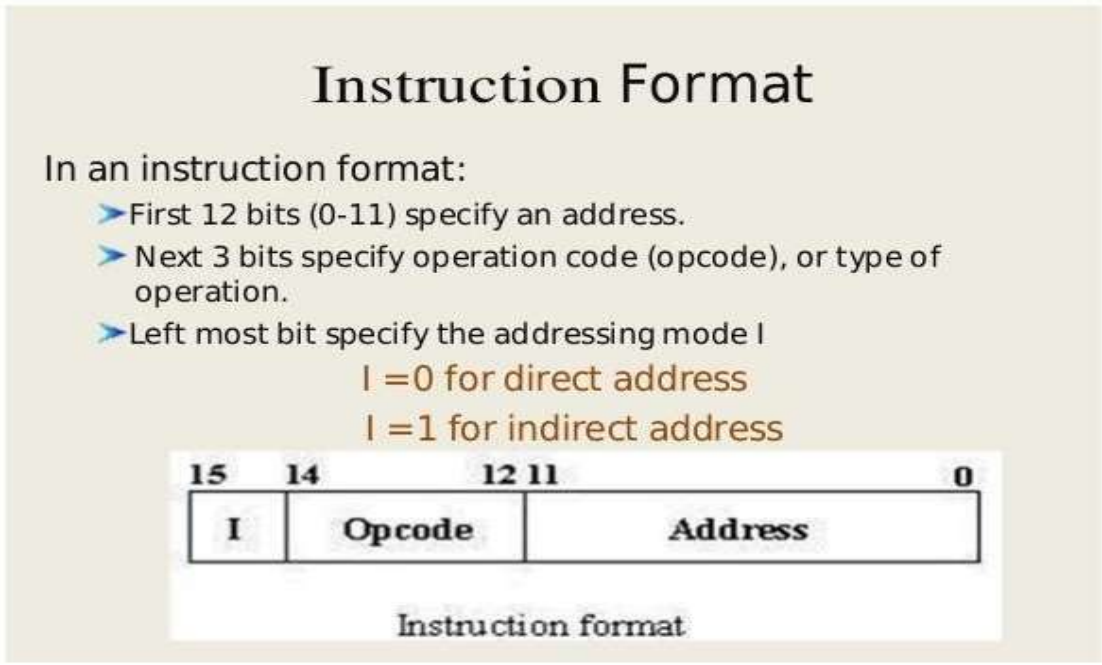
Mode: The way the operand or effective address specified

Ex: ADD R1,R2;

LOAD R1,10;

MOV R2,R1;

SUB M[A],M[B]



Addressing Modes:

The term addressing modes refers to the way in which the operand of an instruction is specified.

The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

The different ways in which a source operand is denoted in an instruction is known as **addressing modes**.

There are different addressing modes in 8086 programming

1. **Immediate addressing mode**

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

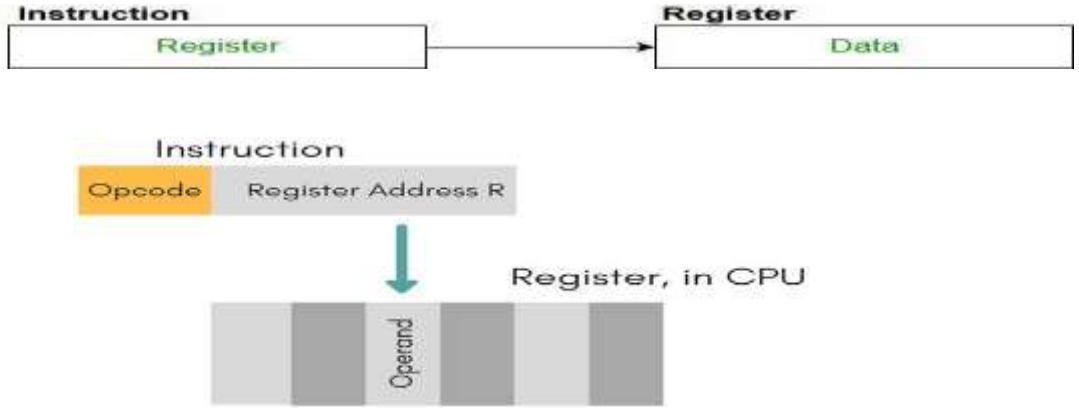
In this mode, the operand is specified in the instruction itself. An immediate mode instruction has an operand field rather than the address field.

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

2. **Register addressing mode**

It means that the register is the source of an operand for an instruction.

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.



Ex: MOV R1, R2 --- Instruction has register R2 and R2 has operand.

Advantages

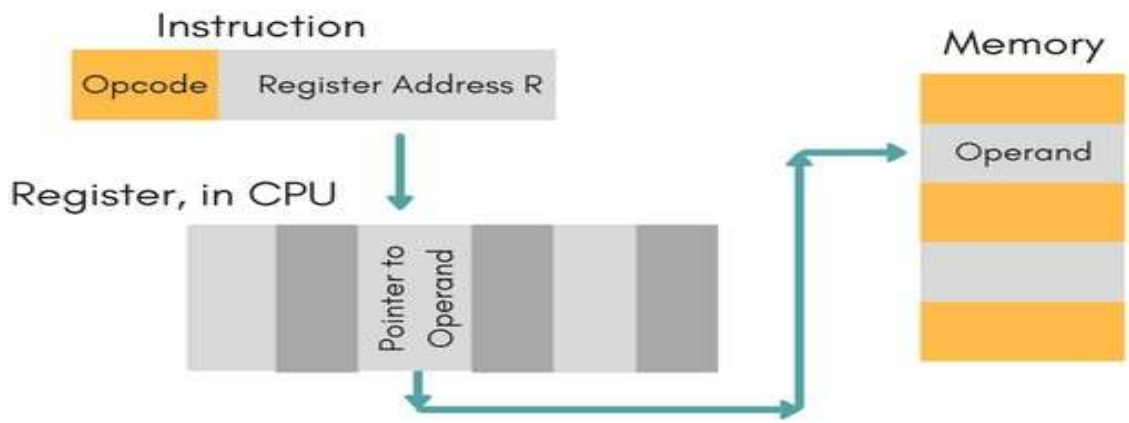
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

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

3. Register Indirect Addressing 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.



Ex: ADD R1, M[R2] or ADD R1, (R2) -- Instruction has register R2 and R2 has memory address of operand.

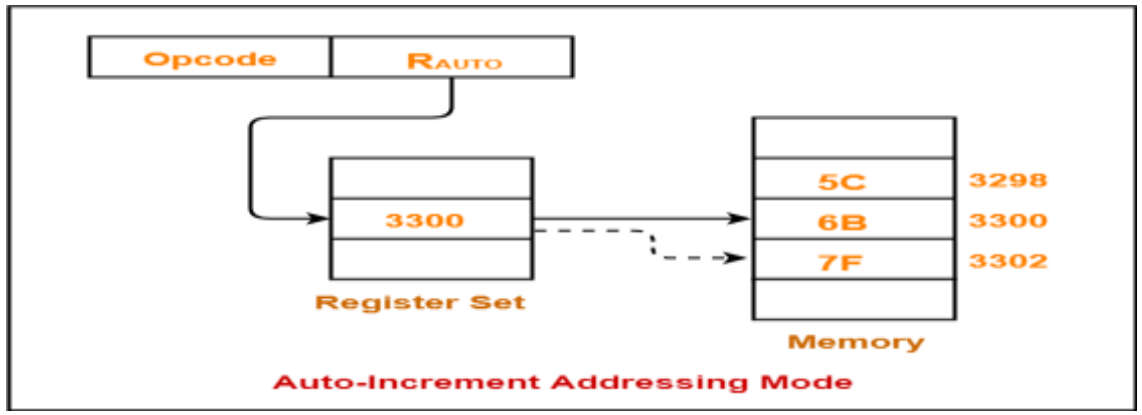
4. Auto Increment/ Decrement Addressing Mode

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

In this addressing mode,

- After accessing the operand, the content of the register is automatically incremented by step size 'd'.
- Step size 'd' depends on the size of operand accessed.
- Only one reference to memory is required to fetch the operand.

Ex: Add R1, (R2) +



Assume operand size = 2 bytes.

Here,

- After fetching the operand 6B, the instruction register R_{AUTO} will be automatically incremented by 2.
- Then, updated value of R_{AUTO} will be 3300 + 2 = 3302.
- At memory address 3302, the next operand will be found (7F).

Auto Decrement addressing Mode:

Effective Address of the Operand = Content of Register – Step Size

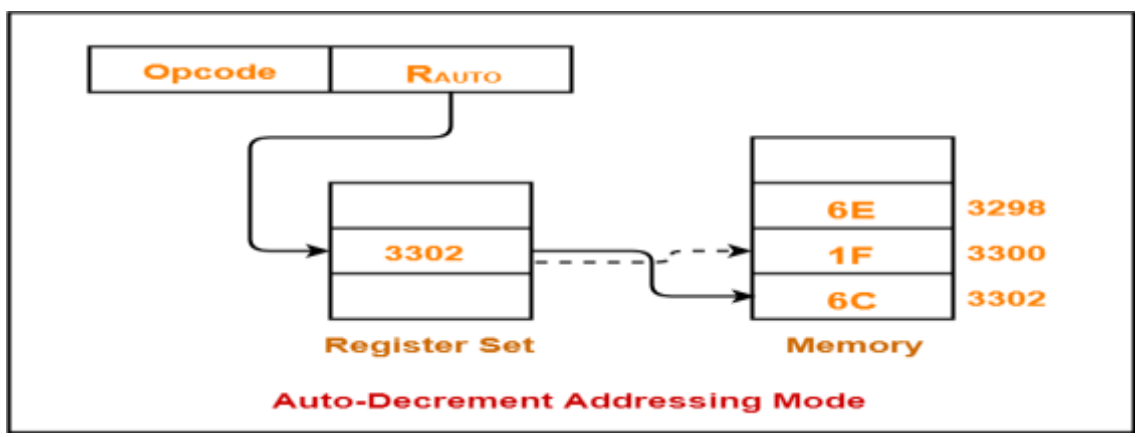
This addressing mode is again a special case of Register Indirect Addressing Mode.

Ex: Add R1, (R2)-

In this addressing mode,

- First, the content of the register is decremented by step size 'd'.
- Step size 'd' depends on the size of operand accessed.
- After decrementing, the operand is read.
- Only one reference to memory is required to fetch the operand.

Example:



Assume operand size = 2 bytes.

Here,

- First, the instruction register R_{AUTO} will be decremented by 2.
- Then, updated value of R_{AUTO} will be 3302 – 2 = 3300.

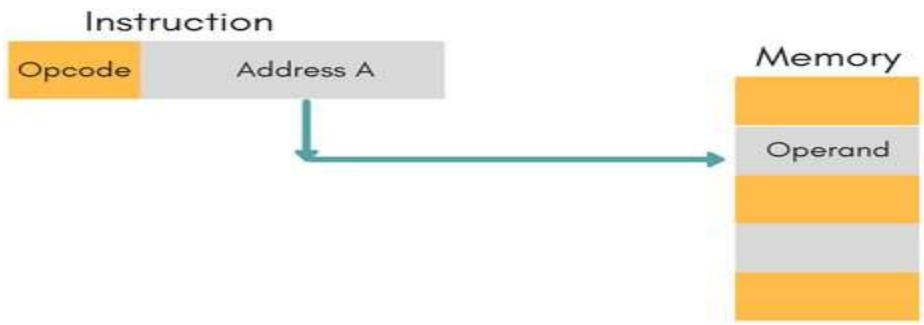
- At memory address 3300, the operand will be found.

5. 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.

Ex2: LOAD R1, 100 Load the content of memory address 100 to register R1.

[100]□30 //30 is a value

6. 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.

Ex: LOAD R1, @100 Load the content of memory address stored at memory address 100 to the register R1.

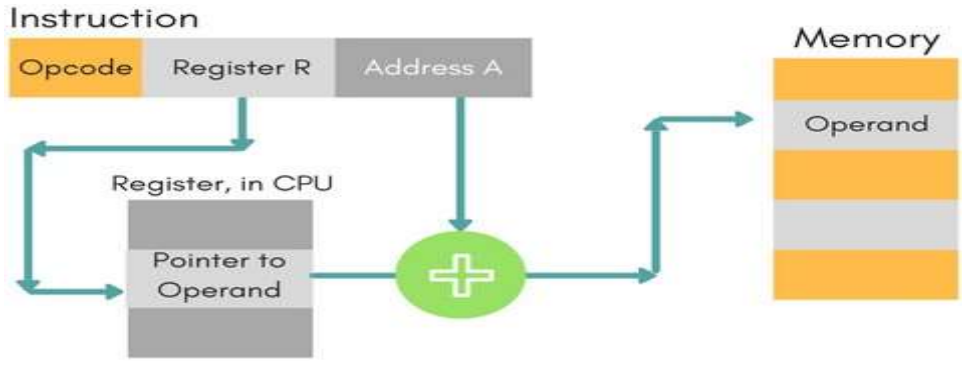
R1 M[100] -----M[200]

M[200]□contains some value #10

7. Displacement Addressing Mode

In this the contents of the indexed register are 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.



Ex: Add R4, 100(R1) R4 <- R4 + M[100+R1]

EA= M[100+R1]

8. Relative Addressing Mode

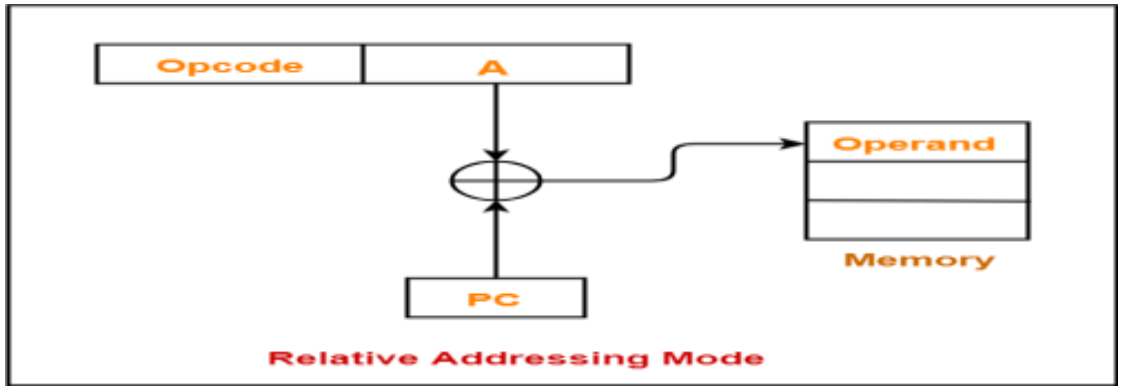
It is a version of Displacement addressing mode.

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

Effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

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

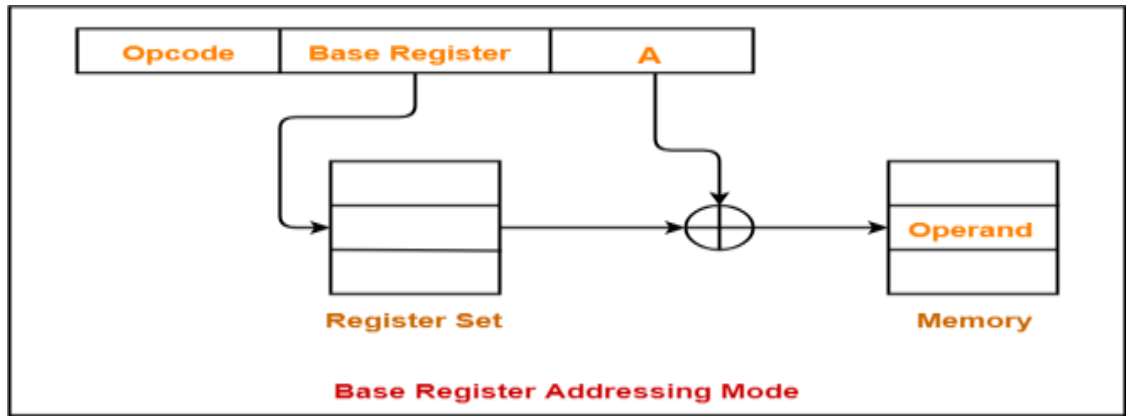
The operand is A cells away from the current cell (the one pointed to by PC)



9. 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.

Effective Address = Content of Base Register + Address part of the instruction



10. 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.

Addressing modes	**Example Instruction**	**Meaning**	**When used**
Register	Add R4, R3	R4 <- R4 + R3	When a value is in a register
Immediate	Add R4, #3	R4 <- R4 + 3	For constants
Displacement	Add R4, 100(R1)	R4 <- R4 + M[100+R1]	Accessing local variables
Register deferred	Add R4, (R1)	R4 <- R4 + M[R1]	Accessing using a pointer or a computed address
Indexed	Add R3, (R1 + R2)	R3 <- R3 + M[R1+R2]	Useful in array addressing: R1 - base of array R2 - index amount
Direct	Add R1, (1001)	R1 <- R1 + M [1001]	Useful in accessing static data
Memory deferred	Add R1, @(R3)	R1 <- R1 + M[M[R3]]	If R3 is the address of a pointer *p*, then mode yields **p*
Auto-increment	Add R1, (R2)+	R1 <- R1 +M[R2] R2 <- R2 + *d*	Useful for stepping through arrays in a loop. R2 - start of array *d* - size of an element
Auto-decrement	Add R1, -(R2)	R2 <-R2-*d* R1 <- R1 + M[R2]	Same as autoincrement. Both can also be used to implement a stack as push and pop

Scaled	Add R1, 100(R2)[R3]	R1<-R1+M[100+R2+R3*d]	Used to index arrays. May be applied to any base addressing mode in some machines.

Notation:
<- -assignment
M -the name for memory

M[R1] refers to contents of memory location whose address is given by the contents of R1