Yeshwantrao Chavan college of Engineering

Department of Information Technology

Subject :Computer Architecture and Organization

Syllabus

Unit I :Basic Structure of Computer Hardware and Software: Functional Units, Basic Operational Concepts, addressing methods and machine program sequencing : Memory Locations , addressing and encoding of information, Main memory operation . Instruction Format, limitations of Short word- length machines, High level language considerations.

Unit II Processing Unit: Some fundamental concepts, Single, two, three bus organization, Instruction set architecture of a CPU – registers, instruction execution cycle, RTL interpretation of instruction, Instruction sequencing, addressing modes. Case study – instruction sets of some common CPUs.

Unit III: Hardwired Control : Design Micro-programmed Control: Microinstructions, Grouping of control signals, Micro program sequencing, Micro Instructions with next Address field, Perfecting microinstruction, Emulation, Bit Slices.Case study – design of a simple hypothetical CPU.

Unit IV: Arithmetic: Number Representation, Addition of Positive numbers, Logic Design for fast adders, Addition and Subtraction , Arithmetic and Branching conditions, Multiplications of positive numbers, Signed- Operand multiplication, fast Multiplication, Booth's Algorithm, Integer Division, Floating point numbers and operations.

Syllabus

Unit V: The main Memory: some basic concepts, semiconductor RAM memories, Memory system consideration, semiconductor ROM memories, Multiple module memories and interleaving, Cache Memory, Mapping techniques, Replacement algorithms, write policies Virtual memories, memory management requirements

Unit VI: Computer Peripherals: I/O Devices, I/O device interface, DMA, Interrupt handling Role of interrupts in process state transitions, I/O device interfaces – SCII, USB. Introduction to Pipelining, Throughput and speedup, pipeline hazards Introduction to parallel processors.

BOOKS

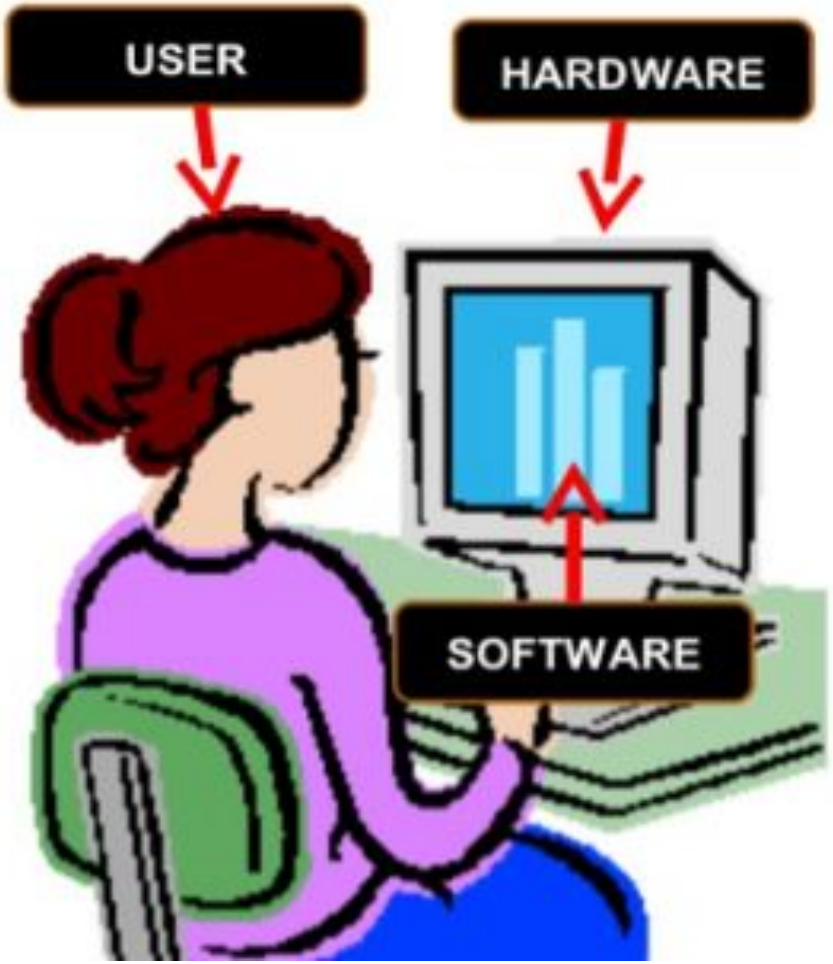
Text Books:

- Carl Hamacher, Zvonko Vranesic, Safwat Zaky: Computer Organization, 5th Edition, Tata McGraw Hill, 2002.
- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Naraig Manjikian : Computer Organization and Embedded Systems, 6 th Edition, Tata McGraw Hill, 2012.

Reference Books:

- 1.William Stallings: Computer Organization & Architecture, 9th Edition, Pearson, 2015.

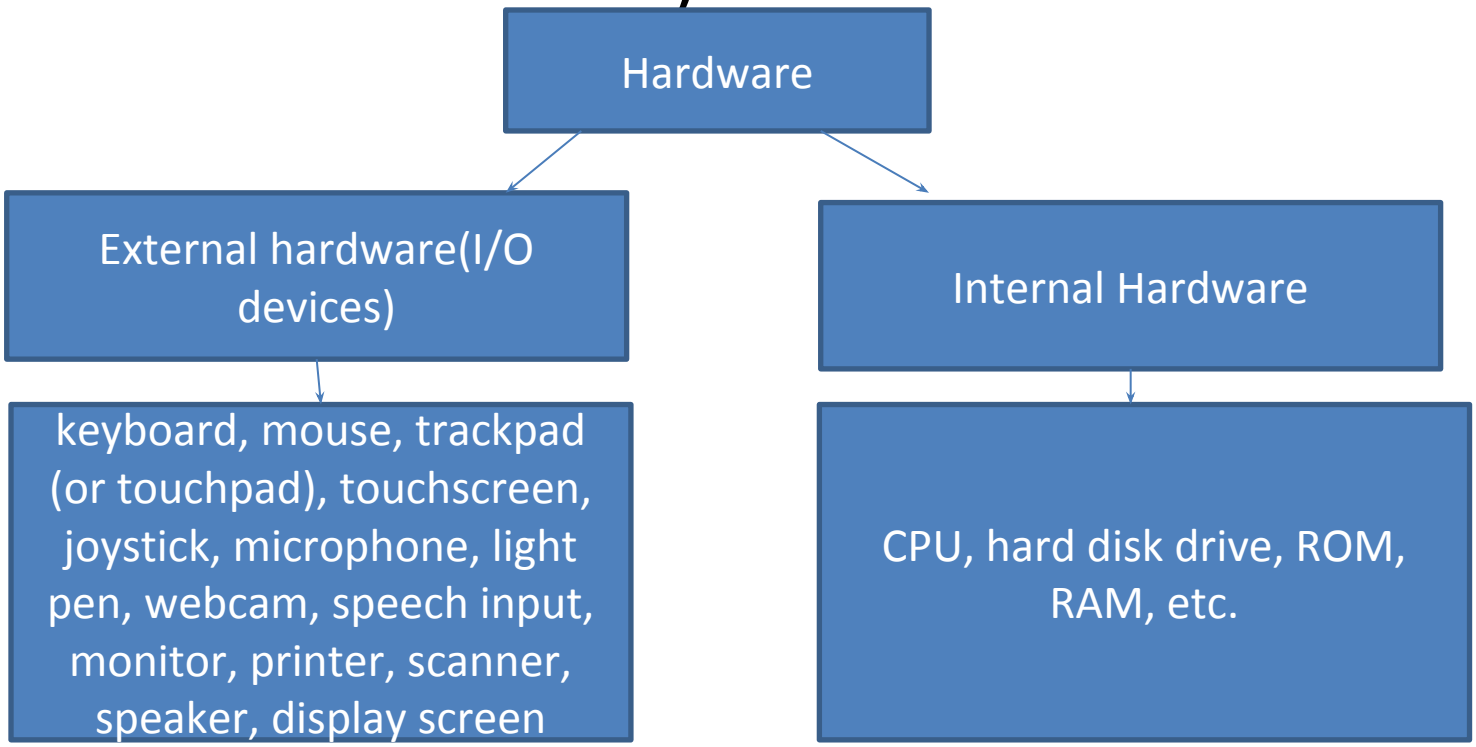
Introduction



- A computer system is defined as **combination of components designed to process data and store files**.
- A computer system requires **hardware, software** and a **user** to fully function.

Hardware and software

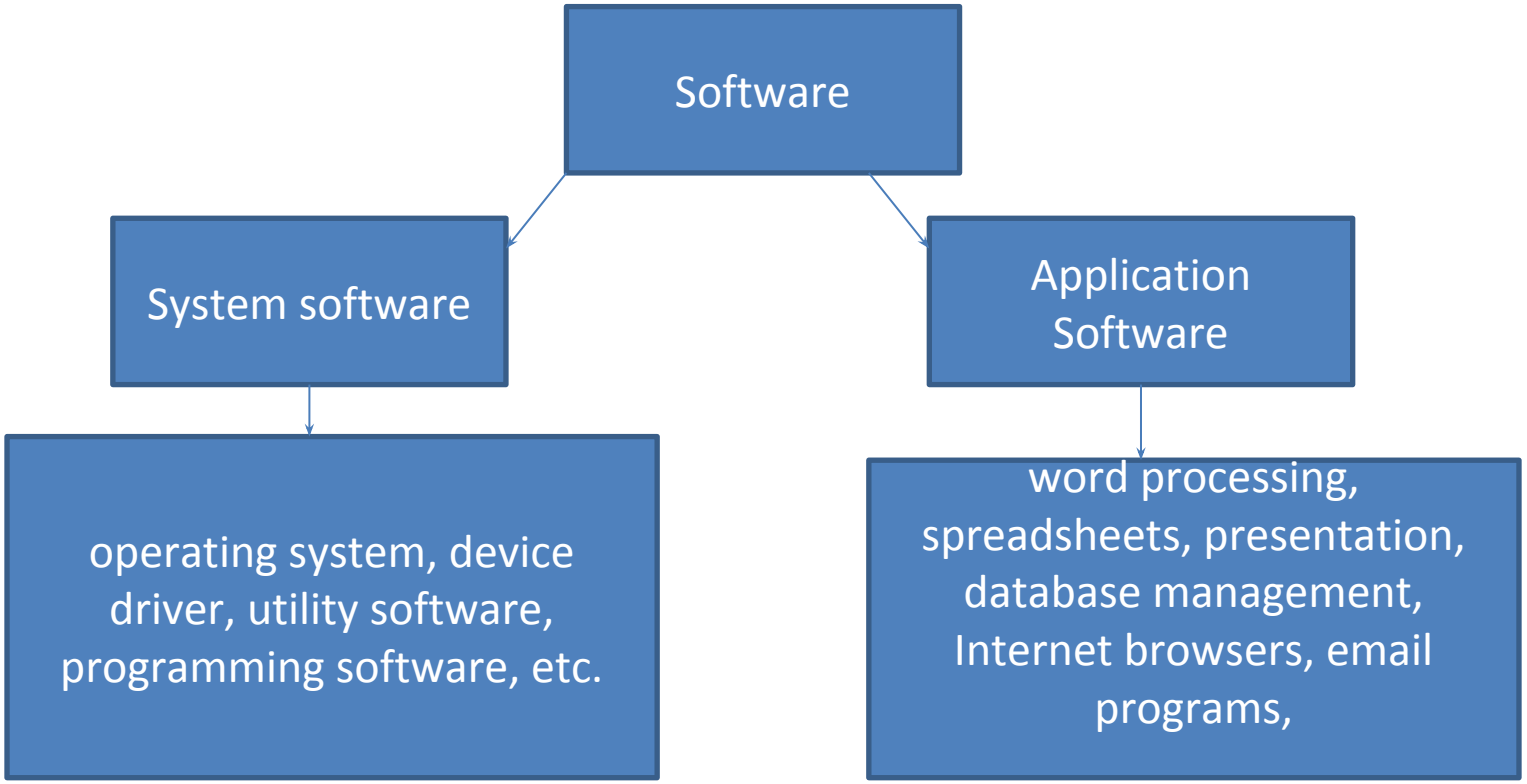
- Hardware: Any physical device or equipment used in or with a computer system (anything you can see and touch).



Hardware and software

- Software:-A set of instructions or programs that tells a computer what to do or how to perform a specific task (computer software runs on hardware).
- Application software: A computer program that provides users with tools to accomplish a specific task.
- System software: It is designed to run a computer's hardware and application software, and make the computer system available for use. It serves as the interface between hardware, application software, and the user.

Software

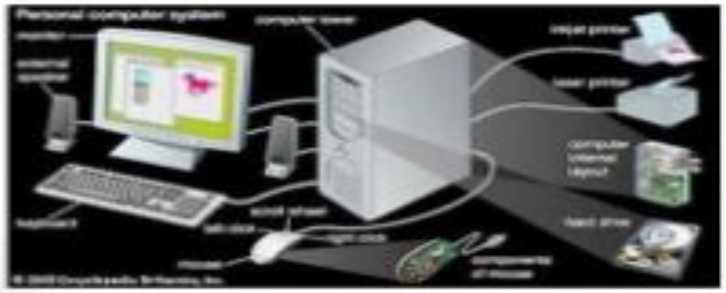

Software
System software
Application Software
operating system, device driver, utility software, programming software, etc.
word processing, spreadsheets, presentation, database management, Internet browsers, email programs,

Hardware and software

Hardware vs Software

Hardware

- CPU/Terminal
- RAM
- Storage unit
- Keyboard, mouse, etc
- Printer



Software

- Programs
- Applications

Definition

- Computer Organization: It is a systematic arrangements ,organization of components of computer.
- Computer Architecture: is the process of flow of data through different components.

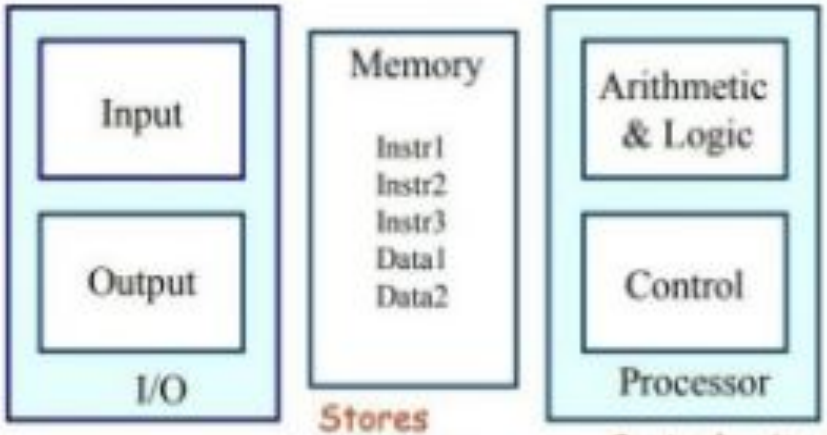
Functional Units of computers

- Every Digital computer systems consist of five distinct functional units.
- 1. Input unit
- 2. Memory unit
- 3. Arithmetic logic unit
- 4. Output unit
- 5. Control Unit

Functional Units

Input unit accepts information:
- Human operators,
- Electromechanical devices (keyboard)
- Other computers

Arithmetic and logic unit(ALU):
- Performs the desired operations on the input information as determined by instructions in the memory



Output unit sends results of processing:
- To a monitor display,
- To a printer

Stores information:
- Instructions,
- Data

Control unit coordinates various actions
- Input,
- Output
- Processing

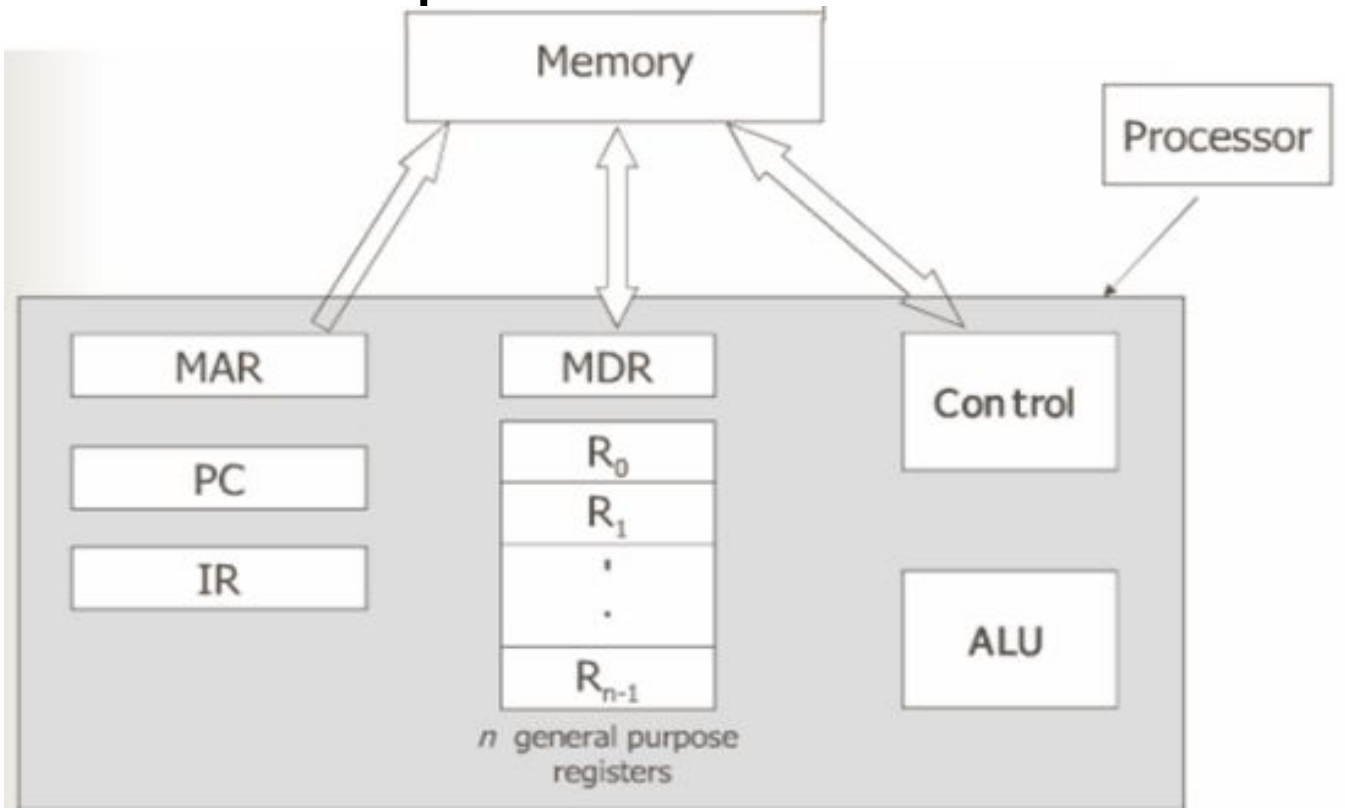
Basic operational concepts

- The program contains of a list of instructions stored in the memory.
- Individual instructions are brought from the memory into the processor, which execute the specified operations.
- Data to be used as operands are also stored in the memory.

Add R1,R2,R3

Basic operational concept

- For execution of this instruction the following registers are required



1.Instruction register (IR):

- The instruction register holds the instruction that is currently being executed.
- Its output is available to the control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction.

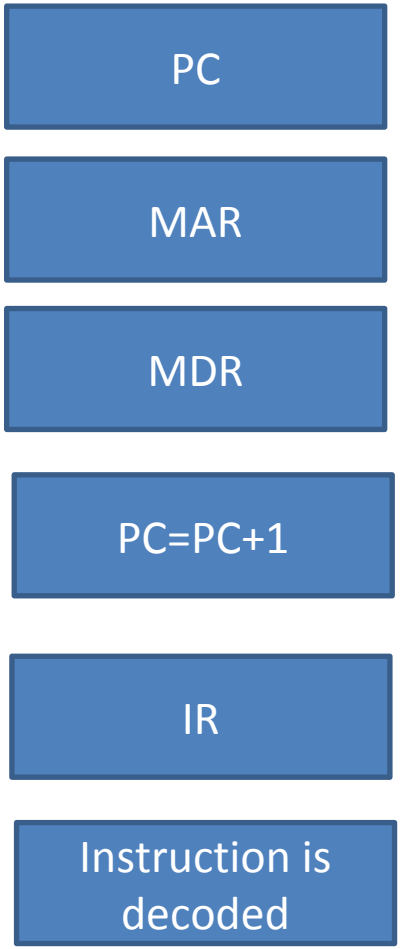
2.Program counter (PC):

- The program counter is another specialized register.
- It keeps track of the execution of a program.
- It contains the memory address of the next instruction to be fetched and executed.
- During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed

3.Memory address register (MAR) & Memory data register(MDR):-

- These two registers facilitate communication with the memory.
- The MAR holds the address of the location to be accessed.
- The MDR contains the data to be written into or read out of the addressed location.

Instruction flow



Operating steps for Program execution (or) Instruction Cycle :

- Execution of the program (stored in memory) starts when the PC is set to point to the first instruction of the program.
- The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory.
- The addressed word is read out of the memory and loaded into the MDR. Next, the contents of the MDR are transferred to the IR. At this point, the instruction is ready to be decoded and executed.
- If the instruction involves an operation to be performed by the ALU, it is necessary to obtain the required operands.
- If an operand resides in memory (it could also be in a general purpose register in the processor), it has to be fetched by sending its address to the MAR and initiating a Read cycle.
- When the operand has been read from the memory into the MDR, it is transferred from the MDR to ALU.
- 7After one or more operands are fetched in this way, the ALU can perform the desired operation. 8. If the result of the operation is to be stored in the memory, then the result is ent to the MDR.
-

Operating steps for Program execution (or) Instruction Cycle :

- The address of the location where the result is to be stored is sent to the MAR, and a write cycle is initiated. 10. At some point during the execution of the current instruction, the contents of the PC are incremented so that the PC pints to the next instruction to be executed.
- Thus, as soon as the execution of the current instruction is completed, a new instruction fetch may be started.
- In addition to transferring data between the memory and the processor, the computer accepts data from input devices and sends data to output devices. Thus, some machine instructions with the ability to handle I/O transfers are provided.

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.

1.Move R1,LOC => R1←[LOC]

2. Add R1, R2, R3 => R1 ←[R2]+[R3]

3.Add #50,R1

Addressing methods

- A computer performs a task based on the instructions provided.

The most common fields are:

- The operation field specifies the operation to be performed, like addition,sub,mul 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.

Addressing methods

- Based on the number of addresses, instructions are classified as:
- Zero address instruction
- One address instruction
- Two address instruction
- Three address instruction

Addressing Methods/Instruction Formats

Three-Address Instructions

- ADD R2, R3, R1 R1 ← [R2] + [R3]

Two-Address Instructions

- ADD R2, R1 R1 ← [R1] + [R2]

One-Address Instructions

- ADD M AC ← [AC] + M[AR]

4. Zero-Address Instructions

- ADD TOS ← [TOS] + [TOS – 1]

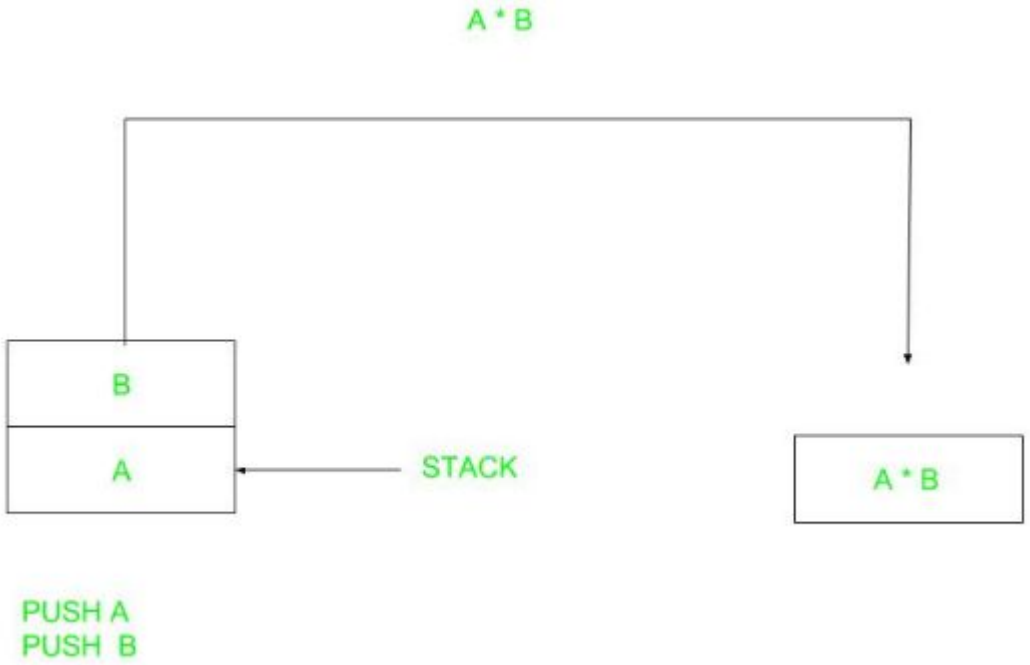
5. RISC Instructions

Lots of registers. Memory is restricted to Load & Store

Addressing Methods/Instruction Formats

- **Zero Address Instructions**
- These instructions do not specify any operands or addresses. Instead, they operate on data stored in registers or memory locations implicitly defined by the instruction.
- For example, a zero-address instruction might simply add the contents of two registers together without specifying the register names.

Addressing Methods/Instruction Formats



To evaluate an expression, it is first converted to reverse Polish Notation i.e. Postfix Notation.

Zero address instruction

```
Expression: X = (A+B)*(C+D)
Postfixed : X = AB+CD+*
TOP means top of stack
```

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]

One address instruction

- These instructions specify one operand or address, which typically refers to a memory location or register. The instruction operates on the contents of that operand, and the result may be stored in the same or a different location.
- 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.

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

- One-Address

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

Two address instruction

- These instructions specify two operands or addresses, which may be memory locations or registers. The instruction operates on the contents of both operands, and the result may be stored in the same or a different location. For example, a two-address instruction might add the contents of two registers together and store the result in one of the registers.
- This is common in commercial computers

Two address instruction

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

Three address Instruction

- These instructions specify three operands or addresses, which may be memory locations or registers. The instruction operates on the contents of all three operands, and the result may be stored in the same or a different location

Example

Instruction Formats

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

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

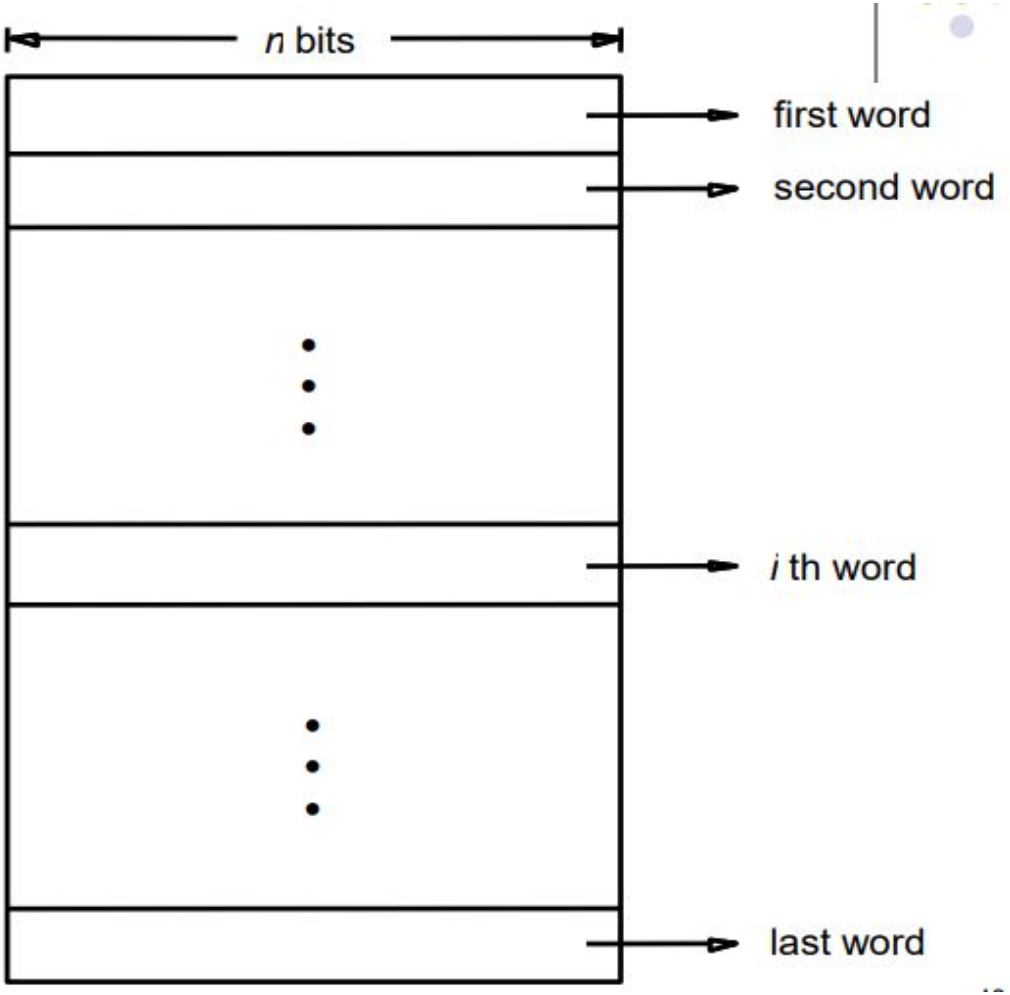
Basics of Memory

- It is used to store data/information and instructions. It is a data storage unit or a data storage device
- Main memory Operations

  - 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

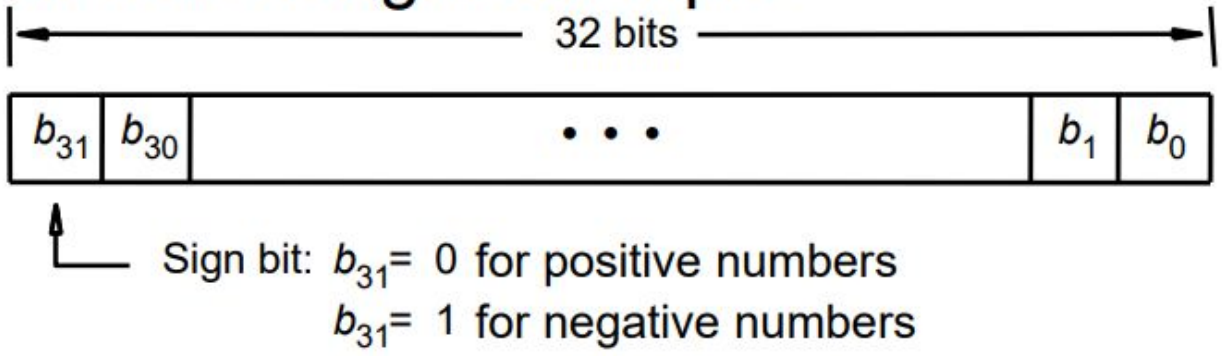
Memory Location, Addresses, and Operation

- Memory consists of many millions of storage cells, each of which can store 1 bit.
- Data is usually accessed in *n*-bit groups. *n* is called word length.



Encoding of Information

- 32-bit word length example



(a) A signed integer

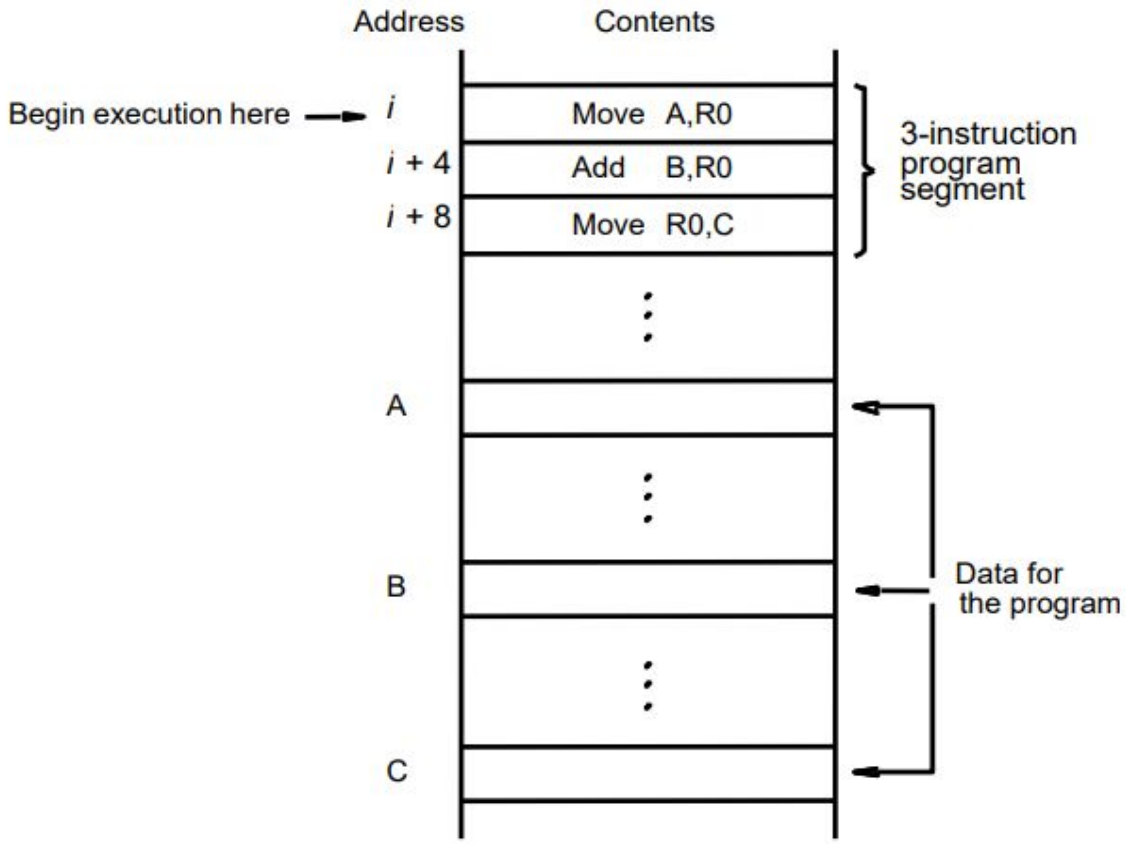
(b) Four characters

Encoding of Information

- If the data is the form of instruction

- To retrieve information from memory, either for one word or one byte (8-bit), addresses for each location are needed.
- A *k*-bit address memory has 2^k memory locations, namely 0 – 2^k-1, called memory space.
- 24-bit memory: 2^{24} = 16,777,216 = 16M (1M=2^{20})
- 32-bit memory: 2^{32} = 4G (1G=2^{30})
- 1K(kilo)=2^{10}
- 1T(tera)=2^{40}

Instruction Execution and Straight-Line Sequencing



Assumptions:
- One memory operand per instruction
- 32-bit word length
- Memory is byte addressable
- Full memory address can be directly specified in a single-word instruction

Two-phase procedure
-Instruction fetch
-Instruction execute

Figure 2.8. A program for C ← [A] + [B].

Branching

Branching

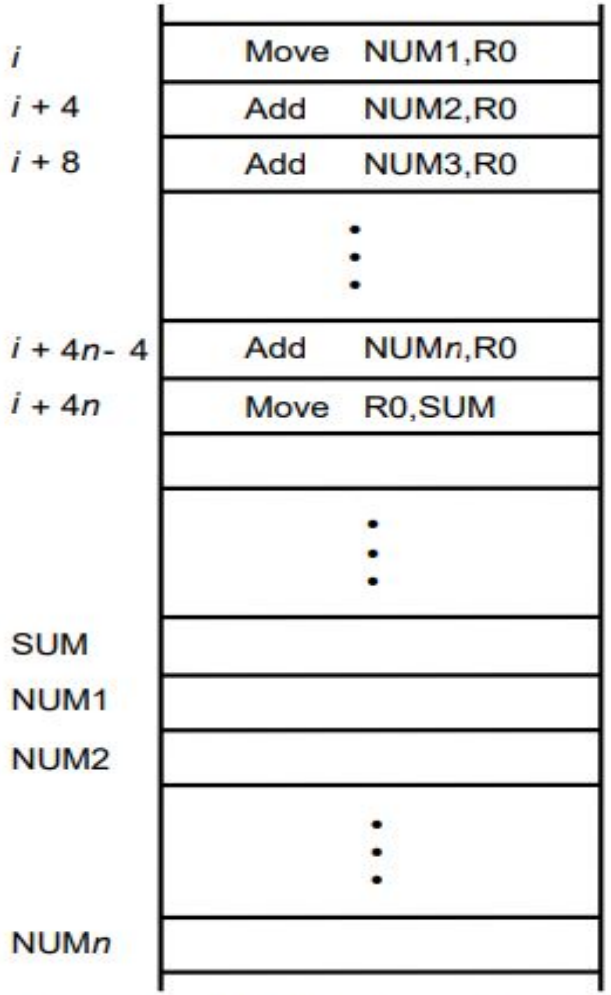


Figure 2.9. A straight-line program for adding *n* numbers.

Branching

- The fundamental idea in programming loop is to cause a straight line sequence to be executed repeatedly.
- If it is determined that branching is taking place the PC is loaded with the address named in the instruction ,otherwise PC is incremented as in usual way.

Branching

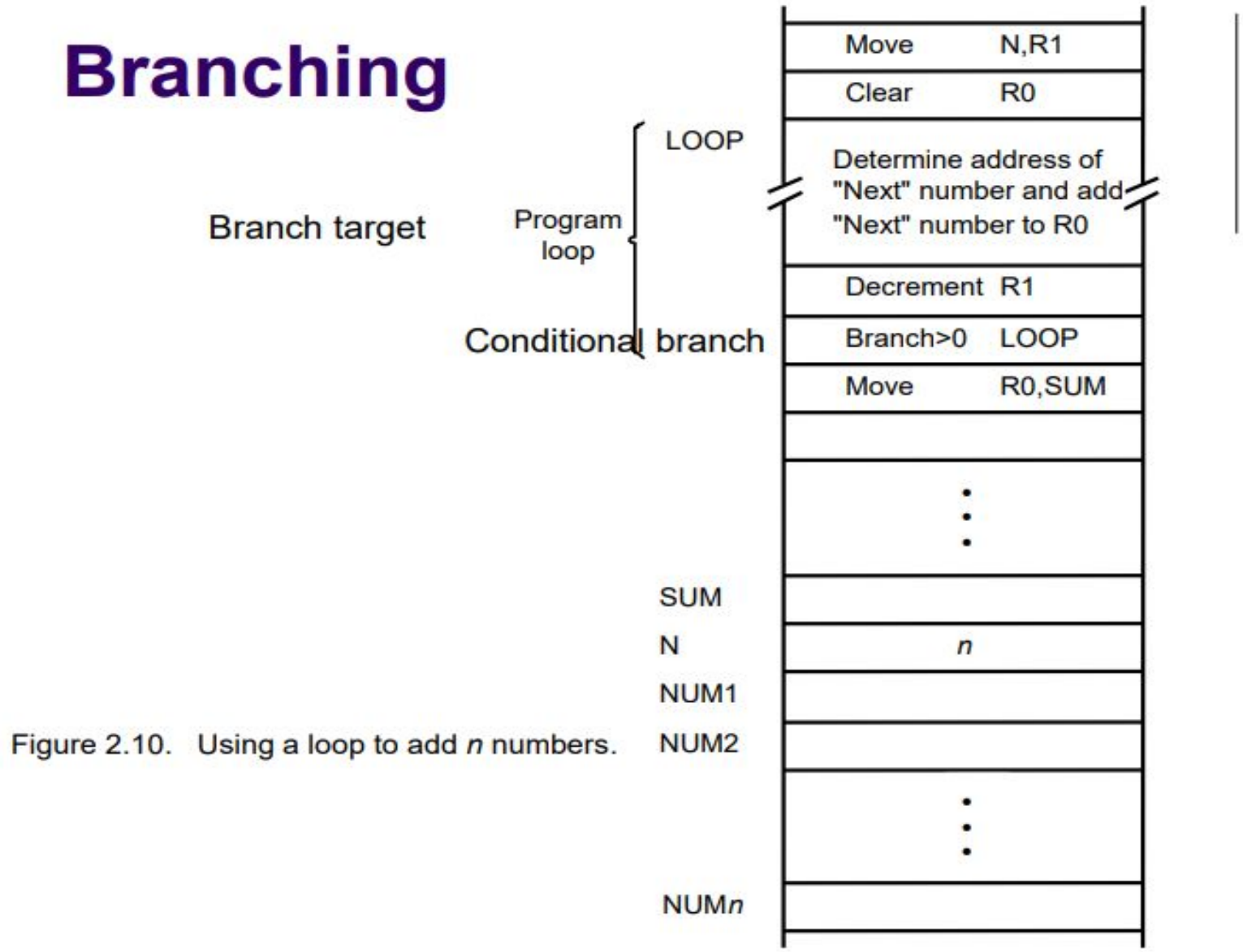


Figure 2.10. Using a loop to add *n* numbers.

Effective address

- **IMPORTANT TERMS**
- **Starting address** of memory segment.
- **Effective address or Offset**: An offset is determined by adding any combination of three address elements: displacement, base and index.
- **Displacement**: It is an 8 bit or 16 bit immediate value given in the instruction.
- **Base**: Contents of base register, BX or BP.
- **Index**: Content of index register SI or DI.
- According to different ways of specifying an operand by 8086 microprocessor, different addressing modes are used by 8086.

Addressing Modes

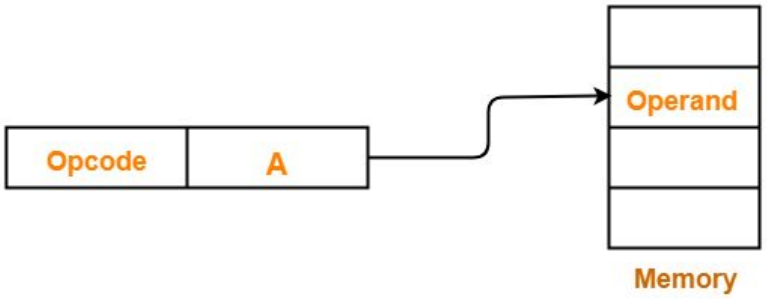
- It refers the way in which the operands of the instruction specified.
- Immediate Addressing Mode
- Direct Addressing Mode
- Indirect Addressing Mode
- Register Direct Addressing Mode
- Register Indirect Addressing Mode
- Relative Addressing Mode
- Indexed Addressing Mode
- Base Register Addressing Mode
- Auto-Increment Addressing Mode
- Auto-Decrement Addressing Mode

Immediate addressing mode

- In this addressing mode,
- The operand is specified in the instruction explicitly.
- Instead of address field, an operand field is present that contains the operand.
- Ex:-MOV R #20 initializes register R to a constant value 20.

Direct addressing mode

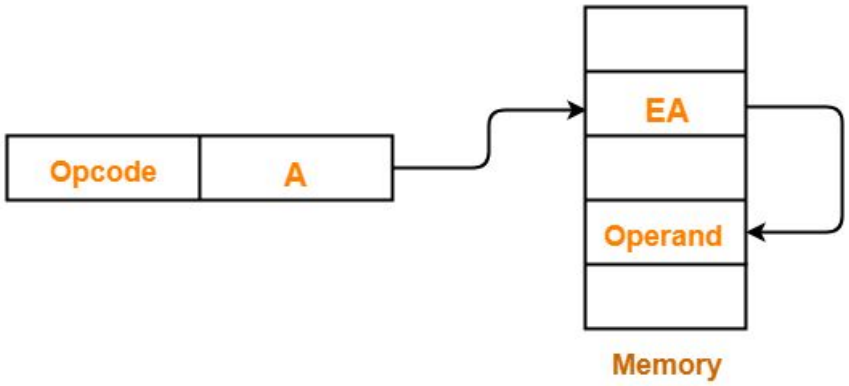
- Effective address of operand is given in the instruction
- Only one reference to memory is required to fetch the operand. Add[5000],r0
- It is also called as **absolute addressing mode**.



Direct Addressing Mode

Indirect Addressing Mode-

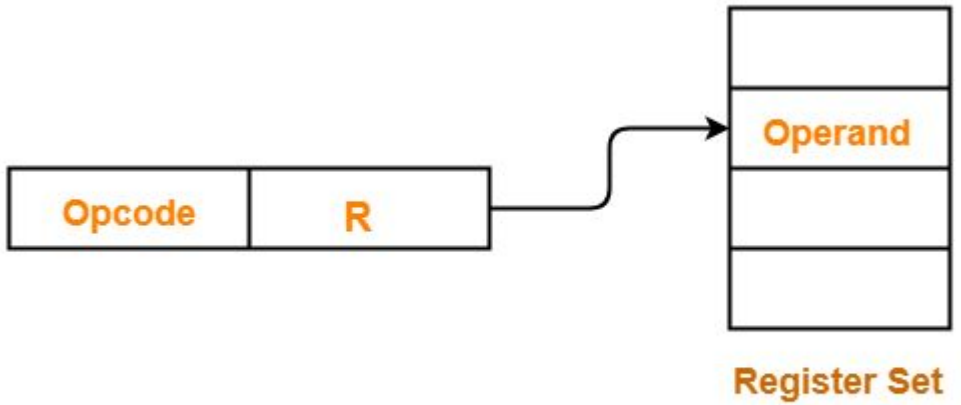
- In this addressing mode,
- The address field of the instruction specifies the address of memory location that contains the effective address of the operand.
- Two references to memory are required to fetch



Indirect Addressing Mode

Register direct addressing mode

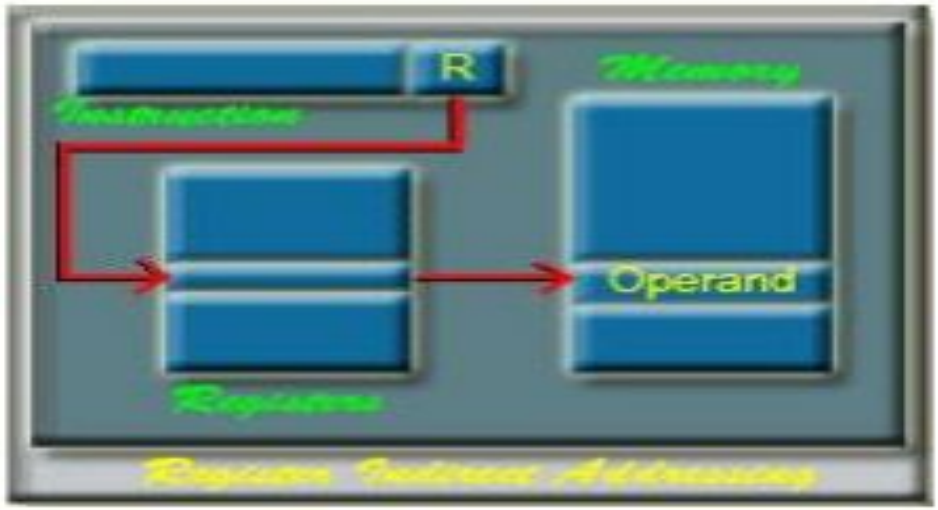
- The operand is present in one of the general purpose register
- MOV A,B



Register Direct Addressing Mode

Register Indirect addressing Mode

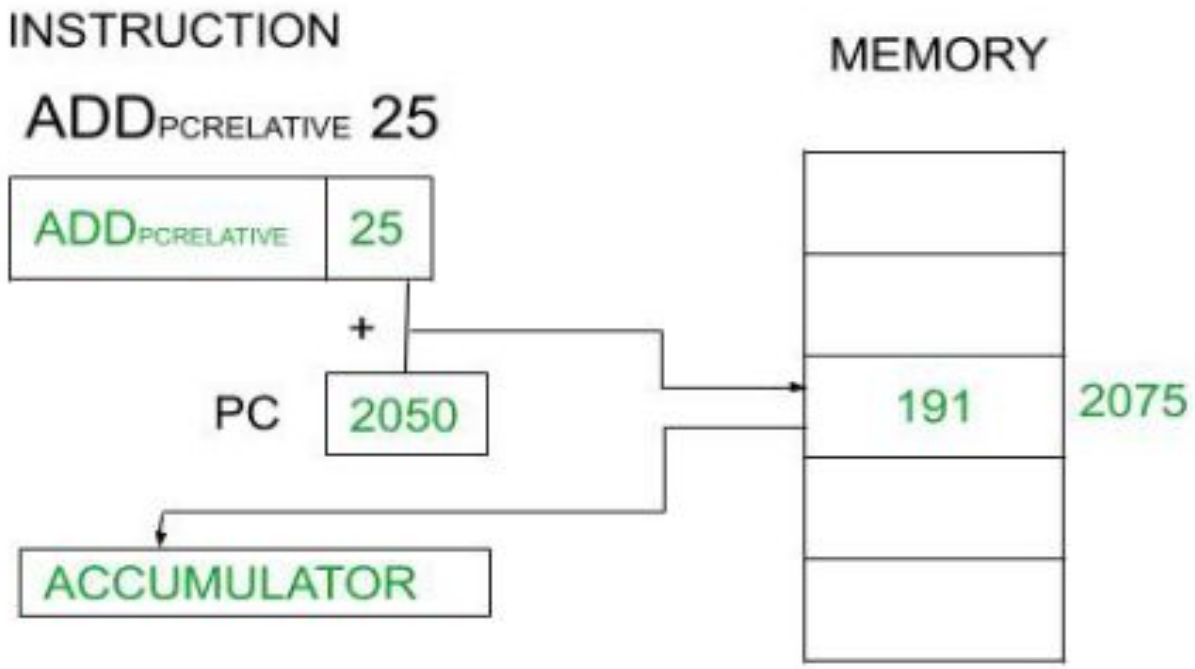
- Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location. It requires only one memory reference and no special calculation. EA = (R)



Relative Addressing mode

- In this mode, the Effective Address (EA) of the operand is calculated by adding the content of the CPU register and the address part of the instruction word.
- The effective address is calculated by adding displacement (immediate value given in the instruction) and the register value.
- The address part of the instruction is usually a signed number, either positive or negative. The effective address thus calculated is relative to the address of the next instruction.

- EA = CPU Register + Displacement



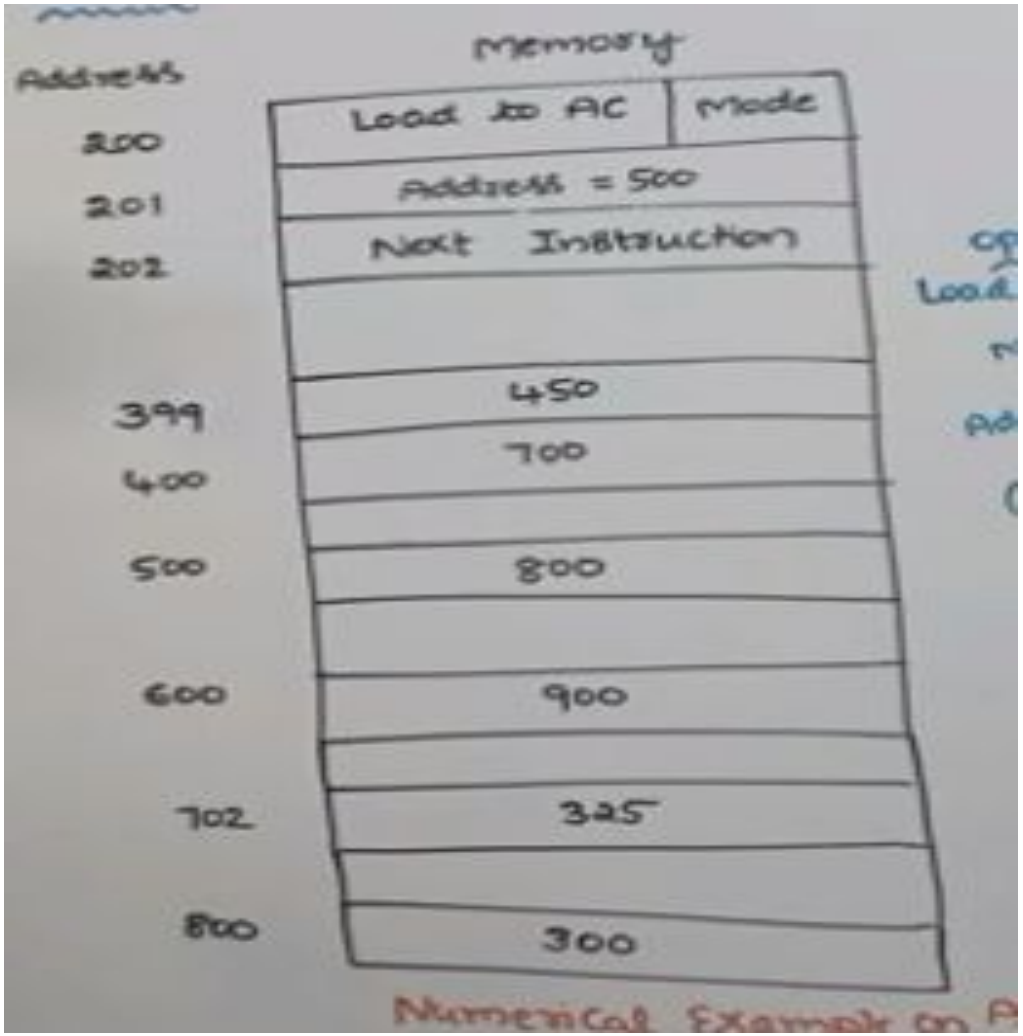
Auto increment addressing mode

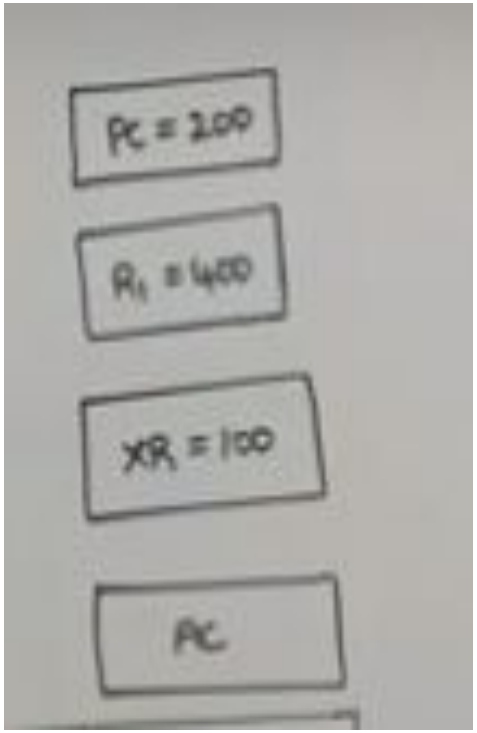
- Effective address of the operand is the contents of the register specified in the instruction .After accessing the operands the contents of this register are automatically incremented to point to the next consecutive memory location.
- ADD R1, (R2)+

Auto decrement addressing mode

- Effective address of the operand is the contents of the register specified in the instruction .Before accessing the operands the contents of this register are automatically decremented to point to the previous consecutive memory location.
- ADD R1, -(R2)

Numerical


Memory
Address
Load to AC
Mode
200
Address = 500
201
Next Instruction
202
450
399
700
400
500
800
600
900
702
325
800
300
Numerical Example on


PC = 200
R1 = 400
XR = 100
AC

Addressing mode	Effective address	Content of AC
Direct Addressing mode	500	800
Immediate Addressing mode	201	500
Indirect Addressing mode	800	300
Relative Addressing mode	702	325
Indexed Addressing mode	600	900
Register Addressing mode	-	400
Register Indirect Addressing mode	400	700
Auto Increment Addressing mode	400	700
Auto decrement AM	399	450

High level language considaration

- A high-level language is any programming language that enables development of a program in a much more user-friendly programming context and is generally independent of the computer's hardware architecture.
- A high-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and registers utilization.
- High-level languages are designed to be used by the human operator or the programmer. They are referred to as "closer to humans." In other words, their programming style and context is easier to learn and implement than low-level languages, and the entire code generally focuses on the specific program to be created.
- A high-level language does not require addressing hardware constraints when developing a program. However, every single program written in a high-level language must be interpreted into machine language before being executed by the computer.
- BASIC, C/C++ and Java are popular examples of high-level languages.

Short word length machines

- The term 'word' is used to describe the number of bits processed at a time by a program or operating system. So, in a 16-bit CPU, the word length is 16 bits. In a 32-bit CPU, the word length is 32 bits.
- Limitations
- **Limited Addressing Range:**
- Reduced Instruction Complexity
- Lower Precision
- Inefficiency for Large Data Types
- **The word length of the machine affects the speed of the machine's operation.**
- **The length of the machine has a great impact on the cost of the hardware**

Information storage units in a computer :-

- 1, bit (bit): is the smallest unit of data in a computer, storing a binary number, that is, 0 or 1. It is also the smallest unit of memory storage information, usually denoted by "B".
- 2, Byte (Byte): bytes are the most common basic unit in a computer that represents storage capacity. A byte consists of a 8-bit binary number, usually denoted by "B". One character occupies one byte, and one Chinese character occupies two bytes. Other common storage units are:

Information storage units in a computer

1KB (kilobyte KB) =1024b

1MB (megabyte megabyte abbreviation "mega") =1024kb

1GB (Gigabyte gigabyte, also known as "gigabit") =1024MB

1TB (Trillionbyte trillion bytes MBytes) =1024gb

1PB (petabyte petabyte Word ticks byte) =1024TB

1EB (Exabyte exascale byte ai byte) =1024pb

1ZB (zettabyte 10 trillion byte ze byte) = 1024x768 EB

1YB (jottabyte 100 million bytes yao byte) = 1024x768 ZB

1BB (brontobyte 100 billion bytes) = YB