UNIT I

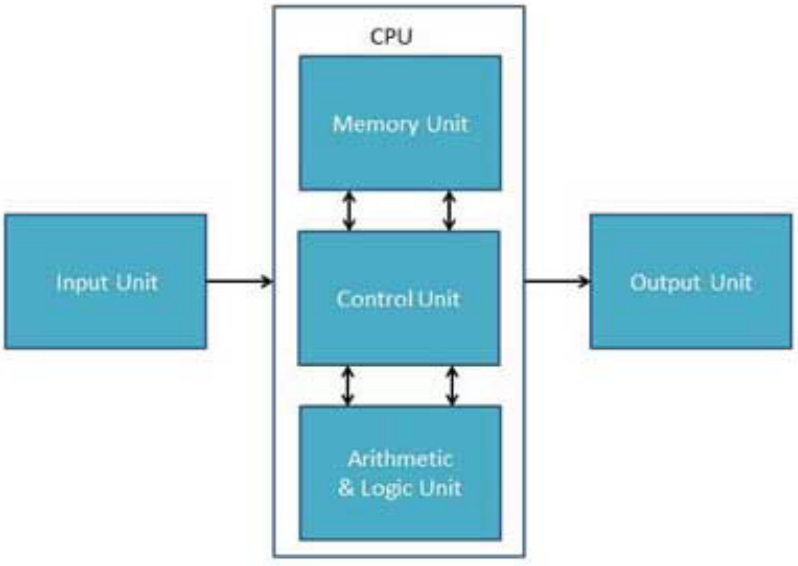
BASIC STRUCTURE OF A COMPUTER SYSTEM

Functional Units – Basic Operational Concepts – Performance – Instructions: Language of the Computer – Operations, Operands – Instruction representation – Logical operations – decision making – MIPS Addressing.

Topic: 1: Functional Units

All types of computers follow the same basic logical structure and perform the following five basic operations for converting raw input data into information useful to their users.

Sn	Operation	Description
1	Take Input	The process of entering data and instructions into the computer system.
2	Store Data	Saving data and instructions so that they are available for processing as and when required.
3	Processing Data	Performing arithmetic, and logical operations on data in order to convert them into useful information.
4	Output Information	The process of producing useful information or results for the user, such as a printed report or visual display.
5	Control the workflow	Directs the manner and sequence in which all of the above operations are performed.



Input Unit

- ✓ This unit contains devices with the help of which we enter data into the computer.
- ✓ This unit creates a link between the user and the computer.
- ✓ The input devices translate the information into a form understandable by the computer.

CPU (Central Processing Unit)

- ✓ CPU is considered as the brain of the computer.
- ✓ CPU performs all types of data processing operations.
- ✓ It stores data, intermediate results, and instructions (program).
- ✓ It controls the operation of all parts of the computer.
- ✓ CPU itself has the following three components –
    - ALU (Arithmetic Logic Unit)
    - Memory Unit
    - Control Unit

ALU (Arithmetic Logic Unit)

- ✓ This unit consists of two subsections namely,
    - Arithmetic Section
    - Logic Section

Arithmetic Section

- ✓ Function of arithmetic section is to perform arithmetic operations like addition, subtraction, multiplication, and division.
- ✓ All complex operations are done by making repetitive use of the above operations.

Logic Section

- ✓ Function of logic section is to perform logic operations such as comparing, selecting, matching, and merging of data.

Memory or Storage Unit

- ✓ This unit can store instructions, data, and intermediate results.

- ✓ This unit supplies information to other units of the computer when needed.
- ✓ It is also known as internal storage unit or the main memory or the primary storage or Random Access Memory (RAM).

- ✓ Its size affects speed, power, and capability. Primary memory and secondary memory are two types of memories in the computer.
- ✓ Functions of the memory unit are –
  - It stores all the data and the instructions required for processing.
  - It stores intermediate results of processing.
  - It stores the final results of processing before these results are released to an output device.
  - All inputs and outputs are transmitted through the main memory.

Control Unit

- ✓ This unit controls the operations of all parts of the computer but does not carry out any actual data processing operations.
- ✓ Functions of this unit are –
  - It is responsible for controlling the transfer of data and instructions among other units of a computer.
  - It manages and coordinates all the units of the computer.
  - It obtains the instructions from the memory, interprets them, and directs the operation of the computer.
  - It communicates with Input/output devices for transfer of data or results from storage.
  - It does not process or store data.

Output Unit

- ✓ The output unit consists of devices with the help of which we get the information from the computer.

- ✓ This unit is a link between the computer and the users. Output devices translate the computer's output into a form understandable by the users.

Topic: 2 Basic Operational Concepts

- To perform a given task on computer, an appropriate program is to be stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.
- Consider an instruction

  Add LOCA, R0
- This instruction adds operand at memory location LOCA to the operand in a register R0 in the processor and the result get stored in the register R0.
- The original content of LOCA is preserved, whereas the content of R0 is overwritten.
- This instruction requires the following steps
  1. The instruction is fetched from memory into the processor.
  2. The operand at LOCA is fetched and added to the content of R_0.
  3. Resulting sum is stored in register R_0.

- The above add instruction combines a memory access operation with an ALU operation.
- Same can be performed using two instruction sequences

  Load LOCA, R1

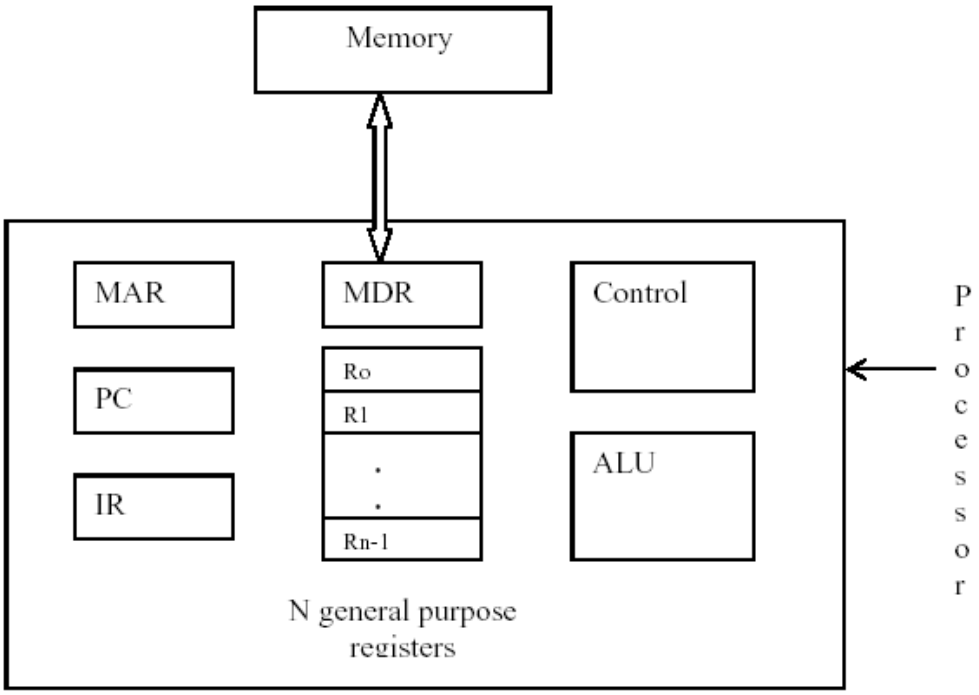
  ADD R1, R0
- Here the first instruction, transfer the contents of memory location LOCA into register R1.
- The second instruction adds the contents of R1 and R0 and places the sum into R0.
- The first instruction destroys the content of R1 and preserve the value of LOCA, the second instruction destroys the content of R0.

Connection between memory and the processor

- Transfer between the memory and the processor are started by sending the address of the memory to be accessed to the memory unit and issuing the appropriate control signals.
- The data are then transferred to or from the memory.

The below figure shows how the memory and the processor can be connected

MAR



- Processor contains number of registers in addition to the ALU and the Control unit for different purposes.
- Various registers are,
  - Instruction register(IR)
  - Program counter(PC)
  - Memory address register(MAR)

- Memory data register(MDR)
- General purpose registers (R0 to Rn-1)

Instruction register (IR):

- IR holds the instruction that is currently being executed by the processor.
- Its output is available to the control circuits, which generates the timing signals that controls various processing elements involved in executing the instruction.

Program counter (PC):

- It is a special purpose register that contains the address of the next instruction to be fetched and executed.
- During the execution of one instruction PC is updated to point the address of the next instruction to be fetched and executed. It keeps track of the execution of a program.

Memory address register (MAR):

- The MAR holds the address of the memory location to be accessed.

Memory data register(MDR):

- The MDR contains the data to be written into or read from the memory location, that is being pointed by MAR.
- These two registers MAR and MDR facilitates communication between memory and the processor.

Operating steps

- Initially program resides in the memory (usually get through the input Unit.) and PC is set to point to the first instruction of the program.
- The contents of PC are transferred to MAR and Read control signal is sent to the memory.

- The addressed word (in this case the first instruction of the program) is read out of the memory and located into the MDR .register.

- Next, the contents of 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 the operands resides in the memory (it could also be in a general purpose register in the processor),
- Then the operands required are fetched from the memory to the MDR by sending its address to the MAR and initiating a read cycle.
- The fetched operands are then transferred to the ALU.
- After one or more operands are fetched dint his way the ALU can perform the desired operation.
- If the result of this operation is to be stored in the memory, then the result is sent to MDR and address of the memory location where the result is to be stored is sent to MAR and write cycle is initiated.
- During the execution of current instruction the contents of the PC are incremented to point to next instruction to be executed.
- Thus as soon as the execution of the current instruction is completed, a new instruction fetch may be started.

Note: in addition to transferring data between the memory and the processor, the computer accepts data from input devices and sends data to the output devices. Thus some machine instructions with ability to handle IO transfers are provided.

Topic: 3 Performance

Performance

"Computer performance is the amount of work accomplished by a computer system".

- ✓ Assessing the performance of computers can be quite challenging.
- ✓ The scale and problem of modern soft ware systems, together with the wide range of performance improvement techniques employed by hardware designers, have made performance assessment much more difficult.
- ✓ When trying to choose among different computers, performance is an important attribute.
- ✓ Accurately measuring and comparing different computers is critical to purchasers and therefore to designers.
- ✓ The people selling computers know this as well.
- ✓ Hence, understanding how best to measure performance and the limitations of performance measurements is important in selecting a computer.

Defining Performance

- ✓ When we say one computer has better performance than another, what do we mean?
- ✓ If you were running a program on two different desktop computers, you'd say that the faster one is the desktop computer that gets the job done first.
- ✓ If you were running a datacenter that had several servers running jobs submitted by many users, you'd say that the faster computer was the one that completed the most jobs during a day.
- ✓ As an individual computer user, you are interested in reducing response time—the time between the start and completion of a task—also referred to as execution time.
- ✓ Datacenter managers are oft en interested in increasing throughput or bandwidth—the total amount of work done in a given time. Hence, in most cases, we will need different performance metrics as well as different sets of applications to

benchmark personal mobile devices, which are more focused on response time, versus servers, which are more focused on throughput.

Response Time

- ✓ Response time Also called execution time.
- ✓ "The total time required for the computer to complete a task", including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.

Throughput

- ✓ It is the "number of tasks completed per unit time".
- ✓ Throughput Also called bandwidth.

Execution Time

- ✓ The actual time the CPU spends computing for a specific task.
- ✓ CPU execution time Also called CPU time .

user CPU time : The CPU time spent in a program itself.

system CPU time : The CPU time spent in the operating system performing tasks on behalf of the program.

Instruction Count

Instruction count The number of instructions executed by the program.

Clock Cycles per Instruction

- ✓ which is the average number of clock cycles each instruction takes to execute, is often abbreviated as CPI.
- ✓ Since different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in the program.
- ✓ CPI provides one way of comparing two different implementations of the same instruction set architecture, since the number of instructions executed for a program will, of course, be the same.

- ✓ To maximize performance, we want to minimize response time or execution time for some task.
- ✓ Thus, we can relate performance and execution time for a computer X:

$$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$

This means that for two computers X and Y, if the performance of X is greater than the performance of Y, we have

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$

$$\text{Execution time}_Y > \text{Execution time}_X$$

That is, the execution time on Y is longer than that on X, if X is faster than Y.

In discussing a computer design, we often want to relate the performance of two different computers quantitatively. We will use the phrase "X is *n* times faster than Y"—or equivalently "X is *n* times as fast as Y"—to mean

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

If X is *n* times as fast as Y, then the execution time on Y is *n* times as long as it is on X:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

Relative Performance

If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is *n* times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times as fast as B.

and A is therefore 1.5 times as fast as B.

In the above example, we could also say that computer B is 1.5 times *slower than* computer A, since

$$\frac{\text{Performance}_A}{\text{Performance}_B} = 1.5$$

means that

$$\frac{\text{Performance}_A}{1.5} = \text{Performance}_B$$

Measuring Performance

- ✓ Time is the measure of computer performance: the computer that performs the same amount of work in the least time is the fastest.
- ✓ Program execution time is measured in seconds per program.
- ✓ However, time can be defined in different ways, depending on what we count.
- ✓ The most straightforward definition of time is called wall clock time, response time, or elapsed time.
- ✓ These terms mean the total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead everything.

The Classic CPU Performance Equation

- ✓ We can now write this basic performance equation in terms of instruction count (the number of instructions executed by the program), CPI, and clock cycle time:
- ✓ CPU time Instruction count CPI Clock cycle time.
- ✓ or, since the clock rate is the inverse of clock cycle time:

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

- ✓ These formulas are particularly useful because they separate the three key factors
- ✓ that affect performance.
- ✓ We can use these formulas to compare two different implementations or to evaluate a design alternative if we know its impact on these three parameters.

MIPS (Million instructions per second)

- ✓ A measurement of program execution speed based on the number of millions of instructions.
- ✓ MIPS is computed as the instruction count divided by the product of the execution time and 10^6.

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

Since MIPS is an instruction execution rate, MIPS specifies performance inversely to execution time; faster computers have a higher MIPS rating.

That is, faster computers mean bigger MIPS

$$\text{MIPS} = \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

……..

Topic 4: Instructions

INSTRUCTIONS

Instructions :

- ✓ A segment of code that contains steps that need to be executed by the computer processor.

Instruction Set :

- ✓ A list of all the instructions with all the variants, which a processor can execute.

Elements of the instruction

- ✓ Each instruction of the CPU has specific information fields, which are required to execute it.
- ✓ These information fields of instructions are called elements of instructions.
- ✓ The instruction elements are,

1) Operation code
2) Source operand address
3) Destination operand address

1.Operation code

- ✓ Specifies the operation to be performed.
- ✓ The operation is specified by the binary code.
- ✓ It is otherwise called Opcode.

2.Source Operand Address

- ✓ It directly specify the source operand address.

3.Destination Operand Address

- ✓ It directly specify the destination operand address.

Example : MOVE ALPHA BETA

Here, MOVE is a Opcode

ALPHA is a source operand

BETA is a destination operand

Types of Instructions

✓ According to number of addresses, the instructions are classified into four types,

1) Zero address instructions
2) One address instructions
3) Two address instructions
4) Three address instructions

1.Zero Address Instructions

- ✓ Instruction contains, no address field or address specified implicitly in the instructions.
- ✓ That instruction contain only an Opcode.
- ✓ Example: CLEAR , START, END

2.One Address Instructions

- ✓ The instructions contain an Opcode and only one operand.
- ✓ Example: ADD B

  Adds the content of the Register B with A register, and result is stored in A.

3.Two Address Instructions

- ✓ The instructions contain an Opcode and two operands.
- ✓ First operand is source operand and second one is destination.
- ✓ Example: MOVE A , B

  Content of register A is moved into register B.

4.Three address instructions

- ✓ The instructions contain an Opcode and three operands.
- ✓ Last two operands are source operands and first one is destination.
- ✓ Example: ADD A , B , C

  B and C contents are added and stored in A register.

According to number of operations, the instructions are classified into some more types,

1. Load and Store instructions.
2. Arithmetic Instructions.
3. Comparison Instructions.
4. Jump Instructions.

1.Load and Store instructions

LDA - Load the value to Accumulator.

LDB : load the value to register B from operand

LDX – Load the value to Index Register.

STA – Store the Accumulator content to some variable.

STB : store the value from register B to some operand.

STX – Store the Index register content into some variable.

2.Arithmetic Instructions

ADD – Add the operand value with Accumulator and result is stored in Accumulator.

SUB - Subtract the operand value with Accumulator and result is stored in Accumulator. MUL - Multiply the operand value with Accumulator and result is stored in Accumulator. DIV - Divide the operand value with Accumulator and result is stored in Accumulator.

ADDF, SUBF, MULF, DIVF – Floating point arithmetic instructions.

3.Comparison Instructions

COMP : that instruction compares the value in the register A with another variable.

And sets the condition code CC to indicate if the accumulator value is < or = or >

4.Jump Instructions

JLT : Jump less than

JEQ : Jump equal to

JGT : Jump greater than

✓ These instructions test the setting of CC and jumps accordingly.

Register to register instructions

ADDR : add the two registers contents and the result is stored rightmost register.

SUBR : subtract the two registers contents and the result is stored rightmost register.

MULR : multiply the two registers contents and the result is stored rightmost register.

DIVR : divide the two registers contents and the result is stored rightmost register.

ADDR S , A

S register content and A register content is added and the result is stored A register.

Special supervisor call instruction.

SVC : it is a kind of system call.

Which transfer control to the OS rather than to the user program.

Topic 5: Instructions – Operations & Operands

INSTRUCTION - OPERATIONS

Types of operations

- ✓ Generally type of operations supported by most of the machines can be categorized as follows,
  - Data transfer operations
  - Arithmetic operations
  - Logical operations
  - Conversion operations
  - I/O operations
  - System control operations
  - Transfer of control operations

Data transfer operations

Operation	Description
Move	Transfer word from source to destination
Store	Transfer word from Processor to Memory
Load	Transfer word from Memory to Processor
Clear	Reset the register contents
Set	Make the register content 1
Push	Transfer word from source to top of the stack
Pop	Transfer word from top of the stack to destination

Arithmetic Operations

Operation	Description
Addition	Perform addition of two operands
Subtraction	Perform subtraction of two operands
Multiplication	Perform multiplication of two operands
Division	Perform division of two operands
Absolute	Replace the value with its absolute value
Negate	Change the sign of the operand
Increment	Adds 1 to operand
Decrement	Subtracts 1 from operand

Logical operations

Operation	Description
AND	Performs logical AND
OR	Performs logical OR
NOT	Performs logical NOT
Exclusive-OR	Performs logical XOR
Test	Test the condition flags
Compare	Compare the two operands and set the flag value
Left Shift	Shift the operand to left position
Right Shift	Shift the operand to right position

Conversion Operations

Operation	Description
Translate	Translate values in a section of memory.
Convert	Converts the contents of a word from one form to another.

I/O Operations

Operation	Description
Input (read)	Transfer data from I/O port to Main memory
Output (write)	Transfer data from main memory to I/O port
Start I/O	Initiate I/O operations
Test I/O	Test the status of the operations

Transfer Control System

Operation	Description
Jump	Unconditional transfer, load to specified address
Jump Conditional	Test the condition and transfer the control
Call to subroutine	Store the current address in stack and Jump into specified address
Return	Return back the control specified from stack
Execute	Execute the instruction.
Skip	Increment PC to skip next instruction
Halt	Stops the program execution
Wait	Stop the program up to specified time
No operation	No operation is performed, but program execution is continued.

Topic 6: Logical Operations

LOGICAL OPERATIONS

- ✓ Although the first computers operated on full words, it soon became clear that it was useful to operate on fields of bits within a word or even on individual bits.
- ✓ Examining characters within a word, each of which is stored as 8 bits.
- ✓ It follows that operations were added to programming languages and instruction set architectures to simplify, among other things, the packing and unpacking of bits into words.
- ✓ These instructions are called logical operations.
- ✓ Figure shows logical operations in C, Java, and MIPS.

Logical operations	C operators	Java operators	MIPS instructions
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit NOT	~	~	nor

- ✓ The first class of such operations is called shift s.
- ✓ They move all the bits in a word to the left or right, filling the emptied bits with 0s.
- ✓ They move all the bits in a word to the left or right, filling the emptied bits with 0s. For example, if register $s0 contained.

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_{two} = 9_{ten}$$

and the instruction to shift left by 4 was executed, the new value would be:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_{two} = 144_{ten}$$

AND

- ✓ A logical bit by bit operation with two operands that calculates a 1 only if there is a 1 in both operands.

OR

- ✓ A logical bit by bit operation with two operands that calculates a 1 if there is a 1in either operand.

NOT

- ✓ A logical bit by bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.

NOR

- ✓ A logical bit by bit operation with two operands that calculates the NOT of the OR of the two operands.
- ✓ That is, it calculates a 1 only if there is a 0 in both operands.

- ✓ Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows –

```
a      =  0011 1100
b      =  0000 1101
-----------------
a&b  =  0000 1100
a|b   =  0011 1101
~a    =  1100  0011
```

Topic 8: Addressing Mode

ADDRESSING MODE

"The different ways in which the location of an operand is specified in an instruction" are referred to as addressing modes.

- ✓ To perform any operation, the corresponding instruction is to be given to the microprocessor. In each instruction, programmer has to specify 3 things:
  - Operation to be performed.
  - Address of source of data.
  - Address of destination(Result)

Reason for using Addressing Modes

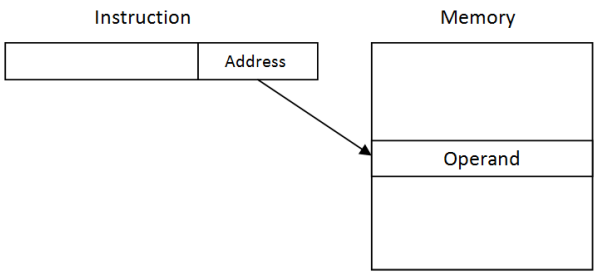
- To reduce the number of bits in the addressing field.
- For program relocation

Types of Addressing Mode

1. Direct addressing mode
2. Indirect addressing mode
3. Immediate addressing mode
4. Index addressing mode
5. Register addressing mode
6. Relative addressing mode
7. Base register addressing mode
8. Auto increment addressing mode
9. Auto decrement addressing mode
10. Stack Addressing mode

1.Direct Addressing Mode (or Absolute Addressing Mode)

The address of the location of the operand is explicitly as a part of the instruction.

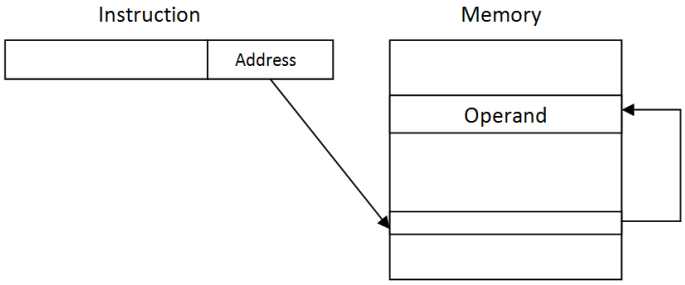


Ex. MOVE 2000, A

✓ This instruction copies the contents of memory location 2000 into register A.

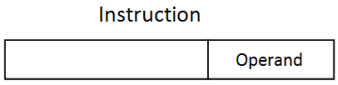
2.Indirect Addressing Mode

In this addressing mode, the instruction contains the address of memory which reefers the address of the operand.



3.Immediate Addressing Mode

The operand is given explicitly in the instruction.



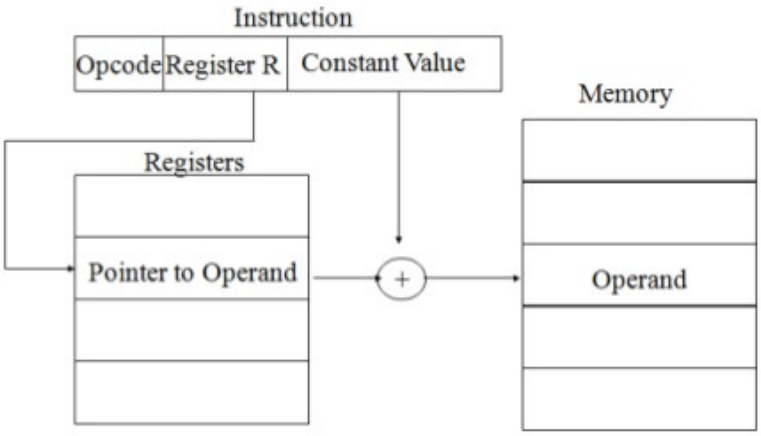
Ex. MOVE #20 , A

✓ The value 20 is copied into register A.

✓ The special symbol # indicates, it is direct value.

4.Index Addressing Mode

✓ The effective address of the operand is generated by adding a constant value to the contents of a register.

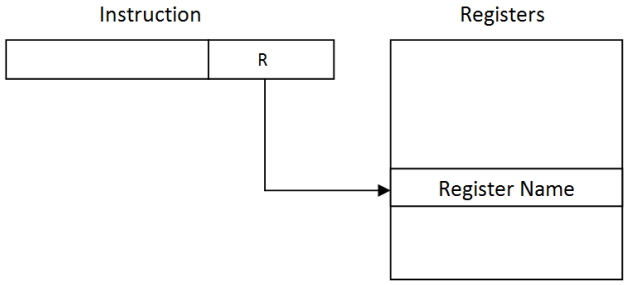


- ✓ The register used may be either a special register provided for this purpose, or, more commonly; it may be anyone of a set of general-purpose registers in the processor.
- ✓ In either case, it is referred to as an index register.
- ✓ We indicate the Index mode symbolically as

EA = Memory address + X

5.Register Addressing Mode

- ✓ The operand is the contents of a processor register; the name (address) of the register is given in the instruction.



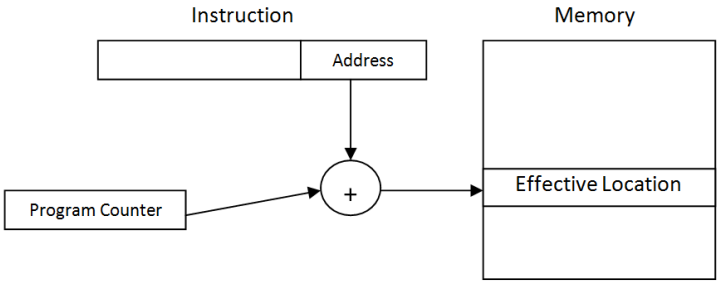
Example: MOVE R1,R2

- ✓ This instruction copies the contents of register R2 to R1.

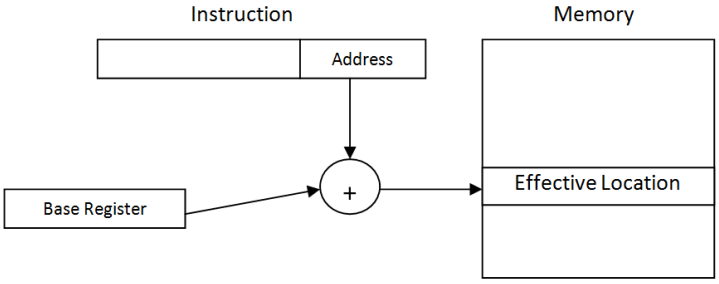
6.Relative Addressing Mode

- ✓ Here Program counter (PC) register is used instead of general purpose register.
- ✓ The effective address is determined by,

EA = Memory address + PC



7.Base Register Addressing Mode



8.Auto Increment Addressing Mode

- ✓ The effective address of the operand is the contents of a register specified in the instruction.
- ✓ After accessing the operand, the content of the register is incremented to address the next location.

Ex. MOV R0 , (R2)+

9.Auto Decrement Addressing Mode

- ✓ The contents of a register specified in the instruction are decremented.
- ✓ And then they are used as an effective address to access a memory location.

Ex. MOV –(R0) , R1

10.Stack Addressing Mode

- ✓ A stack is linear array of reserved memory locations.
- ✓ It is associated with a pointer called Stack Pointer (SP).

- ✓ In this addressing mode, stack pointer always contain the address of Top of Stack, where the operand is to be stored or located.
- ✓ Thus the address of the operand is the content of stack pointer.

Ex. PUSH R

- ✓ This instruction decrements Stack Pointer and copies the contents of register R on top of stack pointed by stack pointer.

********** ALL THE BEST **********