SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTING TECHNOLOGIES

21CSS201T
COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT-II
ARCHITECTURES

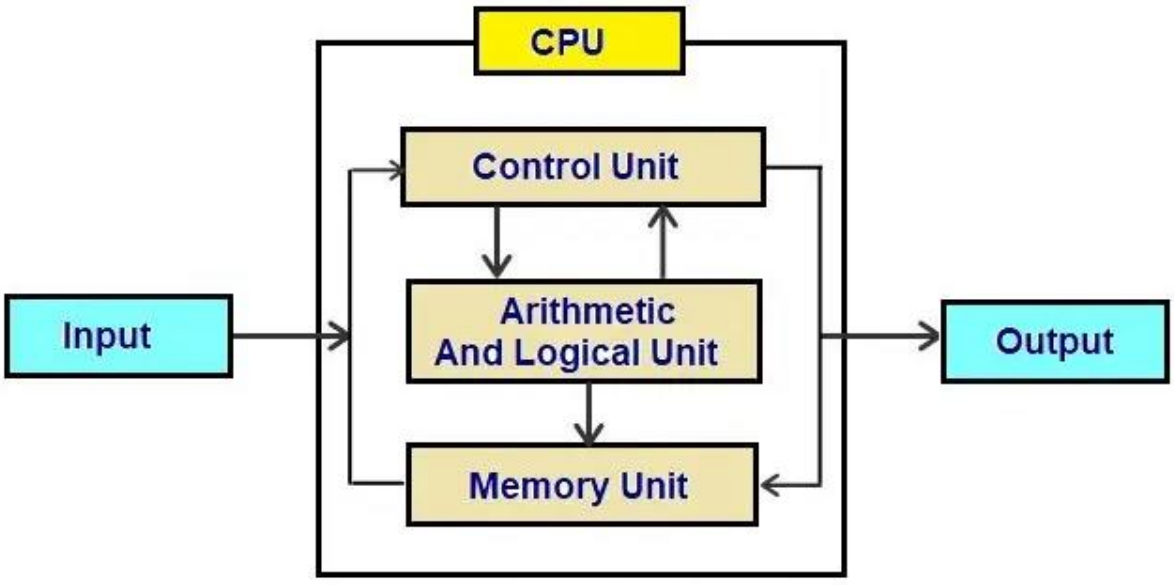
S.PRABU
Assistant Professor
C-TECH-SRM-IST-KTR

Unit-2 – Architectures

Basic structure of computers: Functional Units of a computer, Operational concepts, Bus structures, Memory addresses and operations, assembly language, Instructions, Instruction sequencing, Addressing modes.

FUNCTIONAL UNITS

- ✓ A computer consists of five functionally independent main parts.
- ✓ They are,
  - Input
  - Memory
  - Arithmetic and logic
  - Control unit
  - Output



Input Unit

- ✓ The computer accepts coded information through input unit.
- ✓ The input can be from human operators, electromechanical devices such as keyboards or from other computer over communication lines.
- ✓ The most well known input device is keyboard.

- ✓ Whenever a key is pressed, the corresponding binary code is generated and transmitted over a cable to main memory.
- ✓ The examples of input devices are
  - Keyboard,
  - mouse
  - joysticks,
  - trackballs
  - Microphones

Memory unit

- ✓ Memory unit is used to store programs and data.
- ✓ There are two types of memory,
  - Primary memory
  - Secondary memory

Primary storage

- ✓ It also called main memory.
- ✓ It operates at high speed and it is expensive.
- ✓ It is made up of large number of semiconductor storage cells, each capable of storing one bit of information.

Secondary storage

- ✓ It is slow in speed.
- ✓ It is cheaper than primary memory.
- ✓ Its capacity is high.
- ✓ It is used to store information that is not accessed frequently.
- ✓ Various secondary devices are magnetic tapes and disks, optical disks (CD-ROMs), floppy etc.

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 components
    - ALU (Arithmetic Logic Unit)
    - Control Unit

Arithmetic and Logic Unit

- ✓ Most computer operations are executed in the arithmetic and logic unit(ALU).
- ✓ Consider a typical example,
- ✓ Suppose two numbers located in the memory are to be added.
- ✓ They are brought into the processor, and the actual addition is carried out by the ALU.
- ✓ The sum may then be stored in the memory.
- ✓ Registers
- ✓ When operands are brought into the processor, they are stored in high speed storage elements called registers.
- ✓ Each register can store one word of data.
- ✓ Access time to registers are faster than any other memory system.

Control unit

- ✓ Control unit coordinates the operation of memory, arithmetic and logic unit, input unit, and output unit in some proper way.
- ✓ Control unit sends control signals to other units and senses their states.
- ✓ Data transfers between the processor and the memory are controlled by the control unit through timing signals.
- ✓ Timing signals
- ✓ These are the signals that determine when a given action is to take place.

- ✓ Control units are well defined, physically separate unit that interact with other parts of the machine.
- ✓ A set of control lines carries the signals used for timing and synchronization of events in all units.

Output unit

- ✓ The function of output unit is to produce processed result to the outside world in human understandable form.
- ✓ Examples of output devices are Graphical display, Printers such as inkjet, laser, dot matrix and so on.
- ✓ The operation of a computer can be summarized as follows

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.

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
  - The instruction is fetched from memory into the processor.
  - The operand at LOCA is fetched and added to the content of R_0.
  - 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

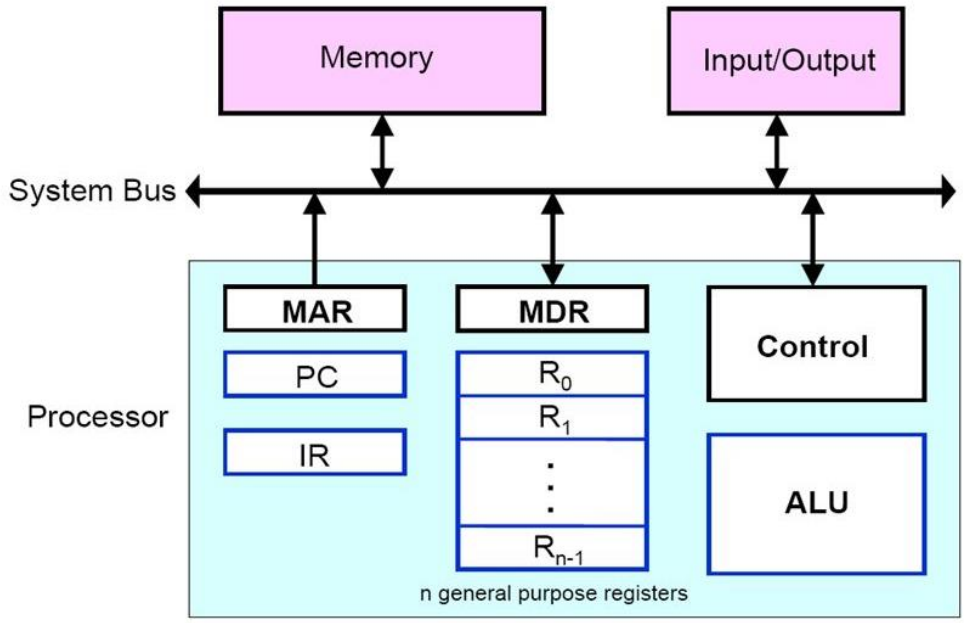
- ✓ Here the first instruction, transfer the contents of memory location LOCA into register R1.
- ✓ The second instruction adds the contents of R1 and R0 and places the sum into R0.
- ✓ The first instruction destroys the content of R1 and preserve the value of LOCA, the second instruction destroys the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



- ✓ 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):

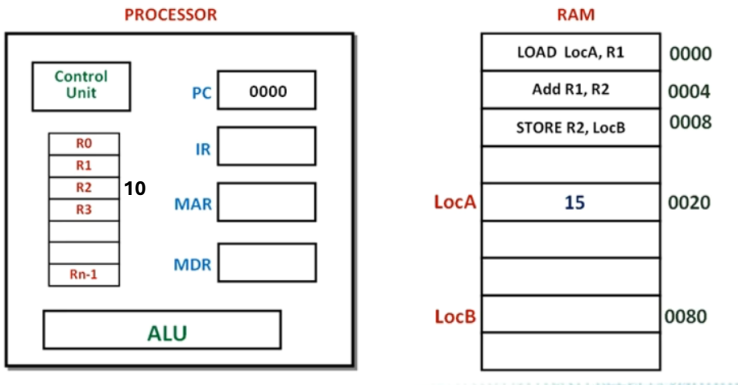
- ✓ The MDR contains the data to be written into or read from the memory location, that is being pointed by MAR.
- ✓ These two registers MAR and MDR facilitates communication between memory and the processor.

Operating steps

- ✓ Consider the following instructions are going to execute,

```
Load  LocA, R1
Add   R1, R2
Store R2, LocB
```

- ✓ Consider the following diagram shows the initial values



Initial Condition.

- ✓ The code(3 instructions) resides in the memory from the address 0000.
- ✓ LocA contains the value 15.
- ✓ R2 register contains the value 10.
- ✓ PC contains the value of the starting address 0000.

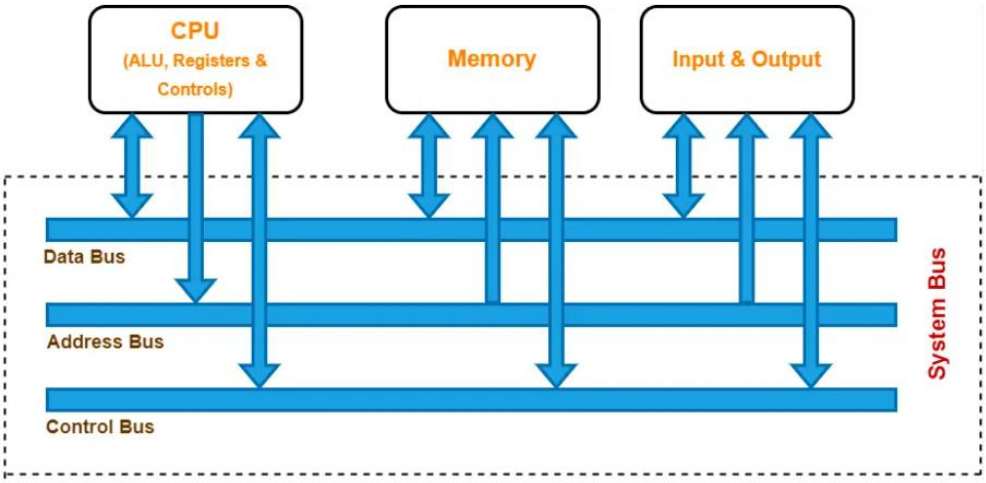
Operating steps

- ✓ First the processor looks at the PC value, now its 0000, that address value can be copied into MAR.
- ✓ Then its value gets updated into second instruction's address, that is 0004.
- ✓ Processor reads the address from MAR, and locate it in RAM, get the value and store it in MDR.
- ✓ MDR content can be moved into IR. Now the instruction fetch is over.
- ✓ The instruction can be decoded by the processor.
- ✓ In that instruction **Load LocA, R1** There is a memory location in RAM is involved.
- ✓ So the execution need Data Fetch also.
- ✓ LocA address (0020) can be stored in MAR.
- ✓ Processor reads that address, and taken the value and stored in MDR.
- ✓ And execute the instruction from IR, the data can be moved into the R1 register.
- ✓ R2 register already contain the value 10, R1 value now is 15.
- ✓ After the first instruction is successfully executed, Processor looks the PC content to execute the next instruction.
- ✓ Its current value is 0004, so that address is shifted into MAR.
- ✓ PC content value updated into next instruction address, that is 0008.
- ✓ The same Instruction fetch can be done in the same manner as first instruction.
- ✓ Then the processor decode the instruction.
- ✓ The instruction **Add R1, R2** contains the both the arguments are registers, which are exist in inside the processor.
- ✓ So no need for data fetch, the processor execute the instruction, the result (25) is stored in R2 register.

- ✓ After the second instruction is successfully executed, Processor looks the PC content to execute the next instruction, that is 0008.
- ✓ After instruction fetch, LocB address 0080 is stored in MAR.
- ✓ Result 25 is moved into MDR.
- ✓ Processor looks the address in MAR, and put the result 25 into the RAM memory location 0080.

BUS STRUCTURES

- ✓ A group of lines serves as a connecting path of several devices is called a BUS.
- ✓ To achieve a reasonable speed of the operation, a computer must be organized so that, all its units can handles one full word of data at a given time.
- ✓ When a word of data is transferred in a bus, all its bits are transferred in parallel, that is, the bits are transferred simultaneously over many wires, or lines, one bit per line.
- ✓ The bus must have a separate line for carrying data, address and control signals.



Data bus : A bus which carries data to and from memory/IO is called as data bus.

Address bus : A bus which carries the address of data in the memory

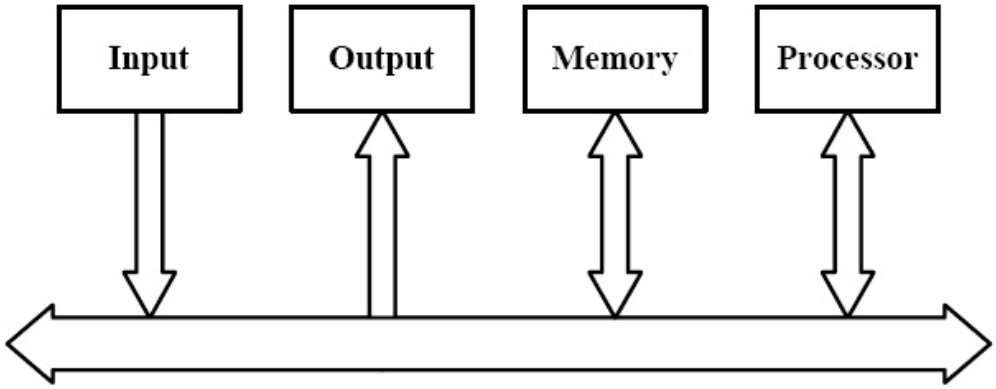
Control Bus : A bus which carries the control signals between the various units of the computer. Ex: Memory Read/write, I/O Read/write

Two types of Bus organizations:

- Single Bus organization
- Multi bus Organization

Single Bus structure

One common bus used to communicate between peripherals and microprocessors.



✓ Single bus is used to interconnect all the units as shown above and hence the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time.

Advantage

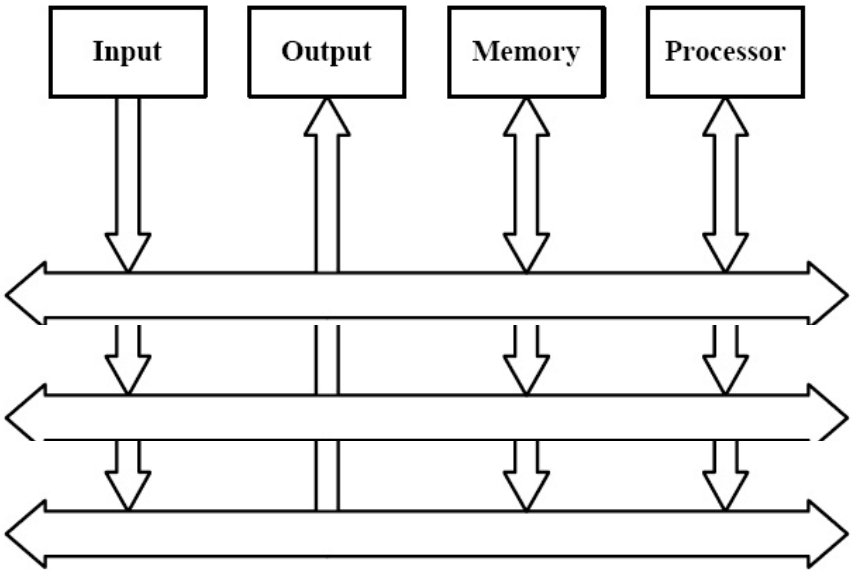
- Simple installation.
- Easy to maintain.

Disadvantage

1. **Limited Bandwidth**: A single bus structure can become a bottleneck in terms of data transfer bandwidth, as all components share the same bus. This can limit the overall performance of the system.
2. **Scalability**: As computer systems become more complex and require higher bandwidth for data transfer, a single bus structure may struggle to scale efficiently.
3. **Contention**: Contention for the bus can occur when multiple components attempt to access it simultaneously, leading to delays and potential performance issues.

Multiple Bus structure

- ✓ System using multiple buses results in concurrency as it allows two or more transfer at the same time.



- ✓ This leads to high performance but at increased cost.

Advantages of Multi-Bus Structure:

1. **Improved Performance**: By segmenting the buses, a multi-bus structure can increase the data transfer bandwidth and reduce contention, improving overall system performance.
2. **Scalability**: Multi-bus architectures are more scalable than single bus architectures. As the system's complexity grows, additional buses can be added to accommodate more devices and higher data transfer rates.
3. **Reduced Bus Contention**: With dedicated buses for specific purposes or peripheral devices, bus contention is minimized, resulting in smoother and more efficient data transfers.
4. **Specialized Communication**: I/O buses allow for specialized communication between the CPU and peripheral devices, optimizing data exchange for specific tasks like graphics rendering or data storage.

Disadvantages of Multi-Bus Structure:

1. **Complexity**: Multi-bus structures are more complex to design and implement than single bus structures, which can increase system cost and complexity.
2. **Higher Hardware Costs**: The use of multiple buses may require additional hardware components, increasing the overall cost of the computer system.

INSTRUCTIONS

Instructions :

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

Ex. Add R1,R2

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

Operation code

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

Source Operand Address

✓ It directly specify the source operand address.

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 based on size

✓ 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.

TYPES OF INSTRUCTIONS BASED ON OPERATION

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

- Load and Store instructions.
- Arithmetic Instructions.
- Comparison Instructions.
- Jump Instructions.
- Register to Register Instructions
- Special purpose Instructions

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.

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.

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 >

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.

Ex. ADDR B , A

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

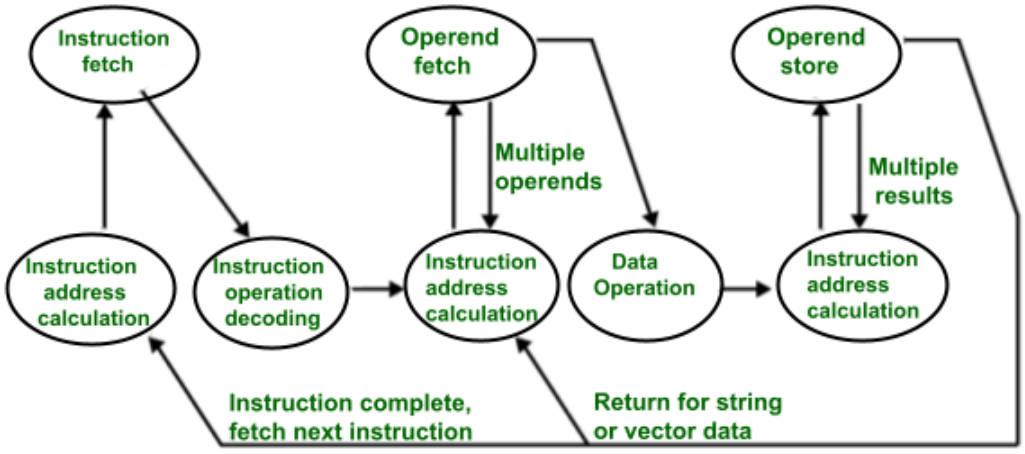
Special supervisor call instruction.

SVC : it is a kind of system call.

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

INSTRUCTION SEQUENCING

- ✓ As instructions are a part of the program which are stored inside the memory.
- ✓ So every time the processor requires to execute an instruction, for that the processor first fetches the first instruction from the memory, then decodes the instruction, fetch the data if necessary then executes the first instruction.
- ✓ Then it fetches the second instruction and so on. The whole process is known as an instruction cycle.



Instruction execution :

Instruction execution needs the following steps, which are

- PC (program counter) register of the processor gives the address of the instruction which needs to be fetched from the memory.
- If the instruction is fetched then, the instruction opcode is decoded. On decoding, the processor identifies the number of operands. If there is any operand to be fetched from the memory, then that operand address is calculated.
- Operands are fetched from the memory. If there is more than one operand, then the operand fetching process may be repeated (i.e. address calculation and fetching operands).
- After this, the data operation is performed on the operands, and a result is generated.
- If the result has to be stored in a register, the instructions end here.
- If the destination is memory, then first the destination address has to be calculated. Then the result is then stored in the memory. If there are multiple

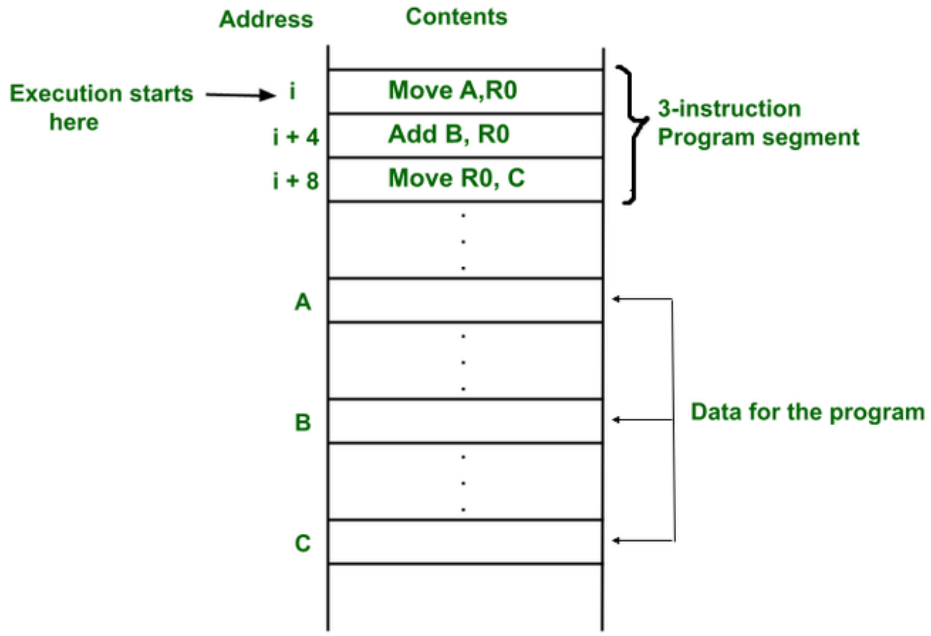
results which need to be stored inside the memory, then this process may repeat (i.e. destination address calculation and store result).

- Now the current instructions have been executed. Side by side, the PC is incremented to calculate the address of the next instruction.
- The above instruction cycle then repeats for further instructions.

Straight line sequencing:

- Straight line sequencing means the instruction of a program is executed in a sequential manner(i.e. every time PC is incremented by a fixed offset).
- And no branch address is loaded on the PC.

Example:



- Here, programs and data are stored in the same memory, i.e. von Neumann architecture.
- First instruction of a program is stored at address i. PC gives address i and instruction stored at that address i is fetched from the memory and then decoded and then operand A is fetched from the memory and stored in a temporary register and then the instruction is executed(i.e. content of address A is copied into processor register R0).

- Side by Side during decoding or execution, the PC gets incremented by 4(i.e. it contains the address of the next instruction) because the instruction and memory segment is of 4 bytes. So the instruction at address i is executed.
- So every time, the PC is incremented by 4. Therefore, the program is executing in a sequential manner. And this process is called straight line sequencing.

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

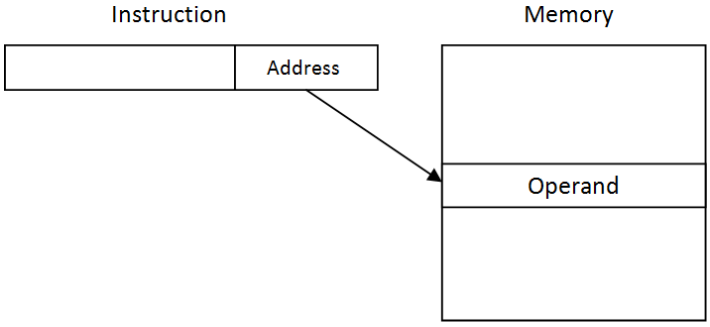
- To reduce the number of bits in the addressing field.
- For program relocation

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. PC Relative addressing mode
7. Base Relative 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.

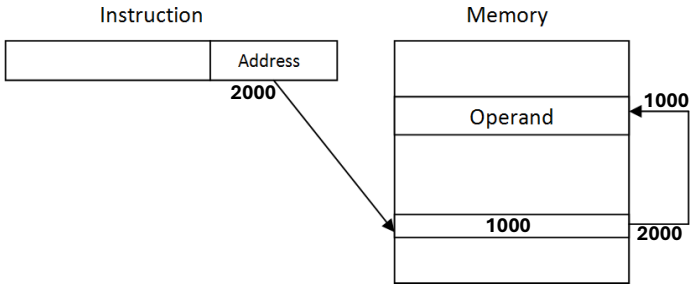


Ex. MOVE 2000, A

- ✓ 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.

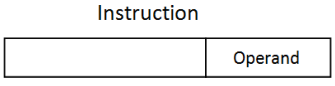


Ex. MOVE [2000], R1

- ✓ The address specified in the instruction is 2000, it is not direct address.
- ✓ It looks the 2000th memory location, read the value, that is 1000.
- ✓ 1000 is the actual address, it reads the content in the 1000th memory location, and copy it into R1.

3. Immediate Addressing Mode

- ✓ The operand is given explicitly in the instruction.

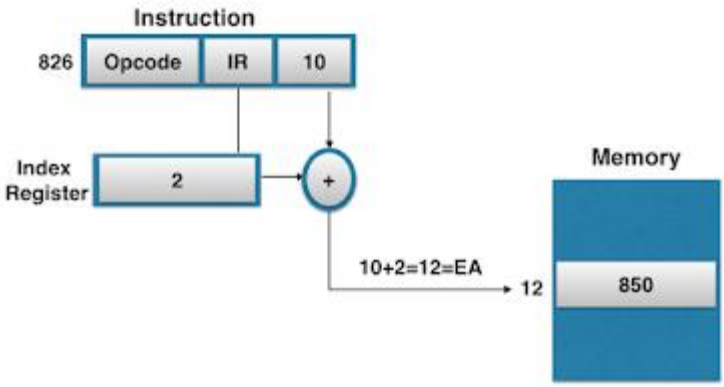


Ex. MOVE #20 , R1

- ✓ The value 20 is copied into register R1.
- ✓ 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.

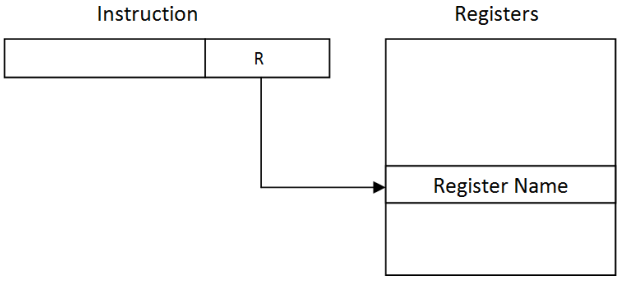


- ✓ The register used may be either a special register provided for this purpose, or, more commonly; it may be anyone of a set of general-purpose registers in the processor.
- ✓ In either case, it is referred to as an index register.
- ✓ We indicate the Index mode symbolically as

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.

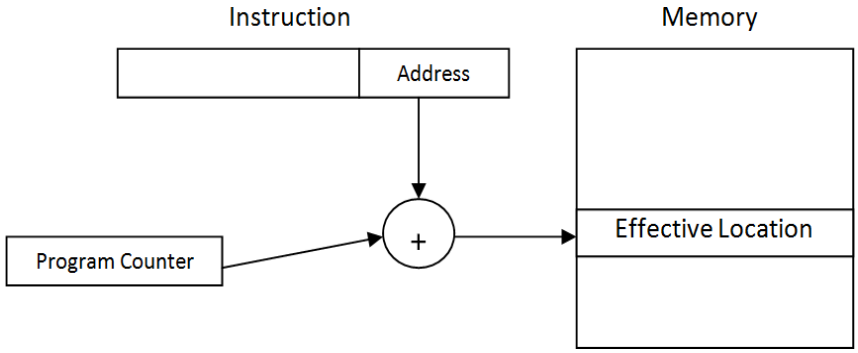


Example: MOVE R1, R2

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

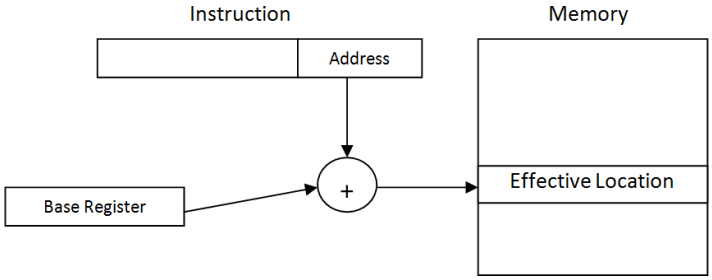
6.PC Relative Addressing Mode

- ✓ Here Program counter (PC) register is used to calculate the effective address instead of normal register.



- ✓ The effective address is determined by, **EA = Memory address + [PC]**
- ✓ When program relocation, PC is using to calculate the Effective address.
- ✓ The PC value can be added with the address specified in the instruction.

7.Base Relative Addressing Mode



Instead of using PC value, It uses Base register content.

The address value present in the instruction, can be taken and added with Base register content.

The effective address is determined by, **EA = Memory address + [B]**

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.