5EC3-01: Computer Architecture

Dr. S. K. Singh, Professor, ECE,
JECRC.

UNIT-I

Syllabus:

- Basic Structure of Computers,
- Functional units,
- Performance issues software,
- Machine instructions and programs,
- Types of instructions,
- Instruction sets: Instruction formats,
- Assembly language,
- Stacks,
- Ques,
- Subroutines.

Introduction:

What are Digital Computers?

The digital computer is a digital system that performs various computational tasks. Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit. Information is represented in digital computers in groups of bits. By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols, such as decimal digits or letters of the alphabet.

Computer architecture

Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program [1].

Computer Organization

Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and interfaces between the computer and peripherals; and the memory technology used [1].

Computer Design

Computer Design is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.

Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as computer implementation. [1].

Basic Structure of Computers

The simplest illustration of a computer shown in figure 1. The computer interacts in some fashion with its external environment. In general, all of its linkages to the external environment can be classified as peripheral devices or communication lines.

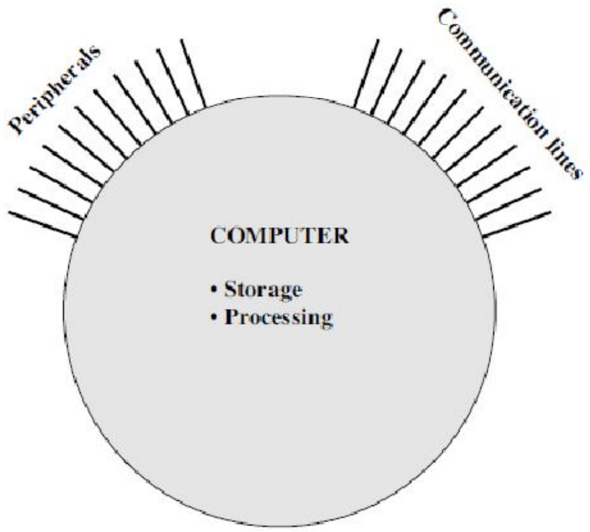


Figure 1. The Computer [1]

The internal structure of the computer itself, which is shown in Figure 2.There are four main structural components:

- **Central processing unit (CPU):** Controls the operation of the computer and performs its data processing functions; often simply referred to as **processor**.
- **Main memory:** Stores data.
- **I/O:** Moves data between the computer and its external environment.
- **System interconnection:** Some mechanism that provides for communication among CPU, main memory, and I/O. A common example of system interconnection is by means of a **system bus**, consisting of a number of conducting wires to which all the other components attach.

Its major structural components are as follows:

- **Control unit:** Controls the operation of the CPU and hence the computer
- **Arithmetic and logic unit (ALU):** Performs the computer's data processing functions
- **Registers:** Provides storage internal to the CPU
- **CPU interconnection:** Some mechanism that provides for communication among the control unit, ALU, and registers

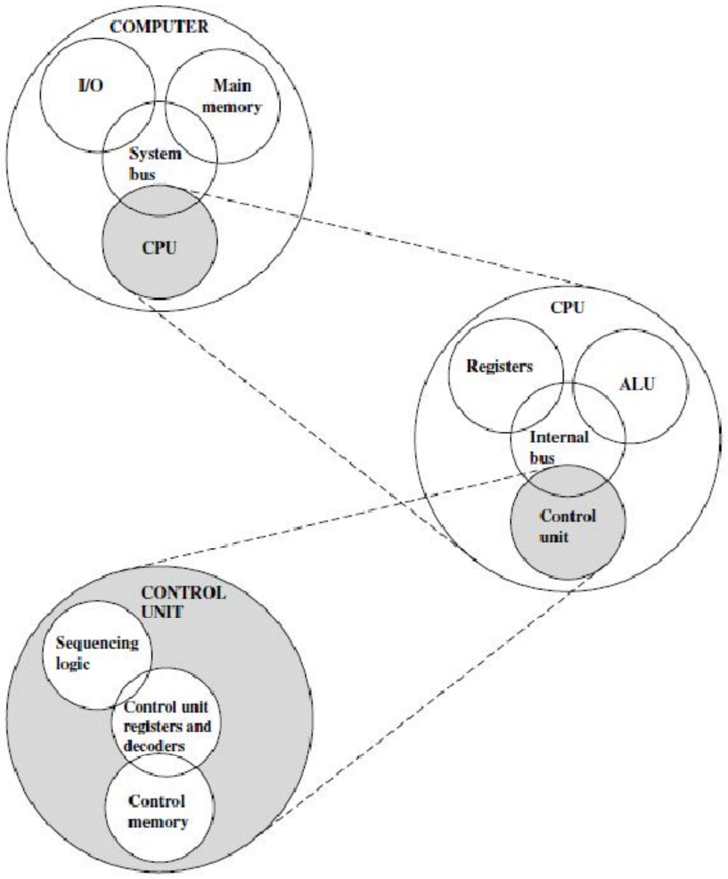


Figure 2 The Computer: Top-Level Structure [1]

Functional units of computer

Figure 3 shows the basic functions that a computer can perform. In general, there are only four:

- Data processing
- Data storage
- Data movement
- Control

- The computer, must be able to **process data**. The data may take a wide variety of forms.

- It is also essential that a computer **store data**. Even if the computer is processing data, the computer must temporarily store at least those pieces of data that are being worked on at any given moment. Files of data are stored on the computer for subsequent retrieval and update.

- The computer must be able to **move data** between itself and the outside world. When data are received from or delivered to a device that is directly connected to the computer, the process is known as *input–output* (I/O), and the device is referred to as a *peripheral.*

- When data are moved over longer distances, to or from a remote device, the process is known as *data communications.*

- Finally, there must be **control** of these three functions. Ultimately, this control is exercised by the individual(s) who provides the computer with instructions.

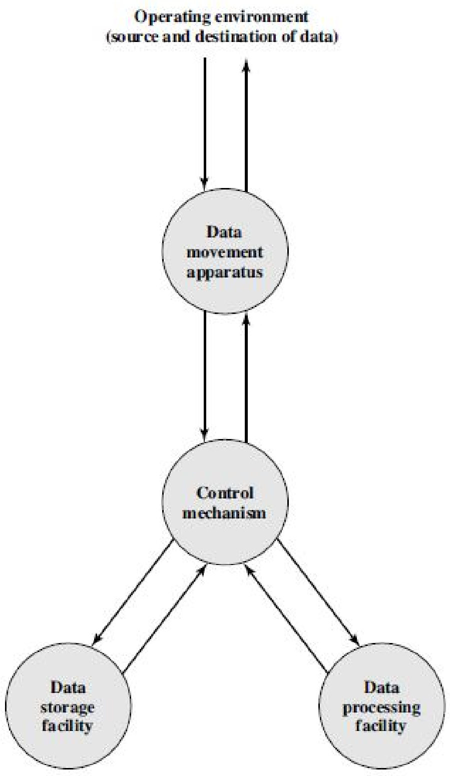


Figure 3. A Functional View of the Computer [1]

Digital Computers: Computer Architecture

Computer Architecture is concerned with the structure and behavior of the computer as seen by the user.

It includes the information, formats, the instruction set, and techniques for addressing memory. The architectural design of a computer system is concerned with the specifications of the various functional modules, such as processors and memories, and structuring them together into a computer system.

Two basic types of computer architecture are:

1. von Neumann architecture[3]
2. Harvard architecture[4]

1. von Neumann architecture

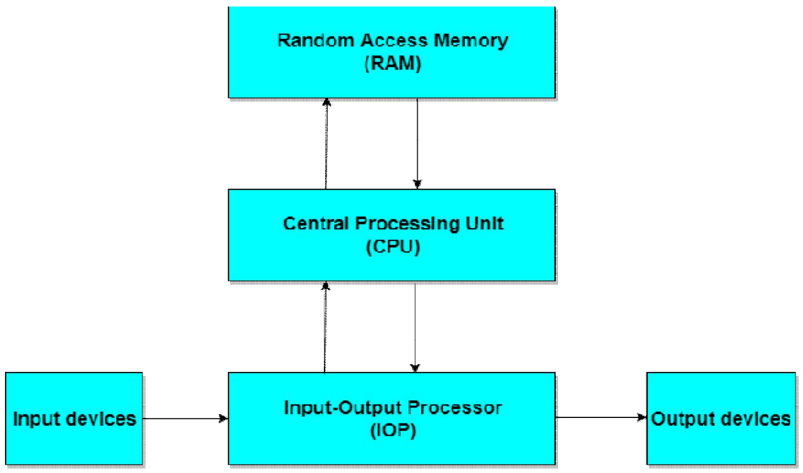
The von Neumann architecture describes a general framework, or structure, that a computer's hardware, programming, and data should follow. Although other structures for computing have been devised and implemented, the vast majority of computers in use today operate according to the von Neumann architecture.

von Neumann envisioned the structure of a computer system as being composed of the following components:

1. ALU: The Arithmetic-Logic unit that performs the computer's computational and logical functions.

2. RAM: Memory; more specifically, the computer's main, or fast, memory, also known as Random Access Memory(RAM).
3. Control Unit: This is a component that directs other components of the computer to perform certain actions, such as directing the fetching of data or instructions from memory to be processed by the ALU; and
4. Man-machine interfaces; i.e. input and output devices, such as keyboard for input and display monitor for output.

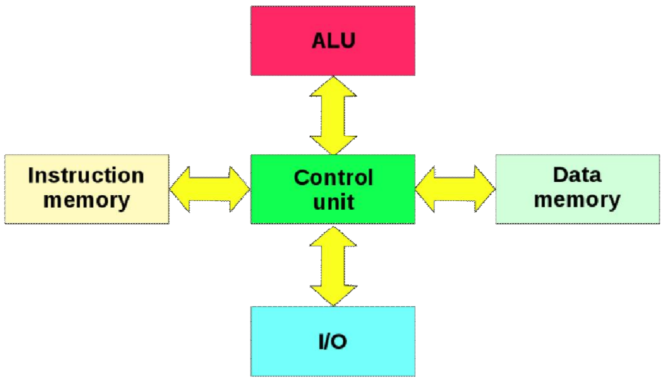
Block diagram of a Digital Computer[3]



An example of computer architecture base on the von Neumann architecture is the desktop personal computer.

2. Harvard architecture

The Harvard architecture uses physically separate storage and signal pathways for their instructions and data[4].



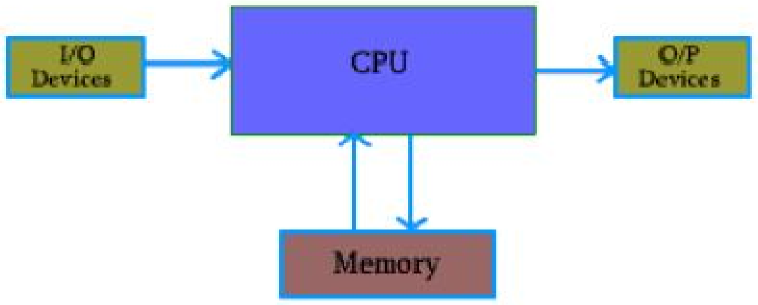
In a computer with Harvard architecture, the CPU can read both an instruction and data from memory at the same time, leading to double the memory bandwidth.

Microcontroller(single-chip microcomputer)-based computer systems and DSP(Digital Signal Processor)-based computer systems are examples of Harvard architecture.

Basic Computer Model and different units of Computer

The model of a computer can be described by four basic units in high level abstraction. These basic units are [2]:

- Central Processor Unit
- Input Unit
- Output Unit
- Memory Unit Input Unit



A. Central Processor Unit [CPU] :

Central processor unit consists of two basic blocks :

- The program control unit has a set of registers and control circuit to generate control signals.
- The execution unit or data processing unit contains a set of registers for storing data and an Arithmatic and Logic Unit (ALU) for execution of arithmatic and logical operations.

In addition, CPU may have some additional registers for temporary storage of data.

B. Input Unit :

With the help of input unit data from outside can be supplied to the computer. Program or data is read into main storage from input device or secondary storage under the control of CPU input instruction. Example of input devices: Keyboard, Mouse, Hard disk, Floppy disk, CD-ROM drive etc.

C. Output Unit :

With the help of output unit computer results can be provided to the user or it can be stored in stograge device permanently for future use. Output data from main storage go to output device under the control of CPU output instructions.

Example of output devices: Printer, Monitor, Plotter, Hard Disk, Floppy Disk etc.

D. Memory Unit :

Memory unit is used to store the data and program. CPU can work with the information stored in memory unit. This memory unit is termed as primary memory or main memory module. These are basically semi conductor memories.

There ate two types of semiconductor memories -

- **Volatile Memory :** RAM (Random Access Memory).
- **Non-Volatile Memory :** ROM (Read only Memory), PROM (Programmable ROM) EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM).

Secondary Memory :

There is another kind of storage device, apart from primary or main memory, which is known as secondary memory. **Secondary memories are non volatile memory** and it is used for permanent storage of data and program.

Example of secondary memories:

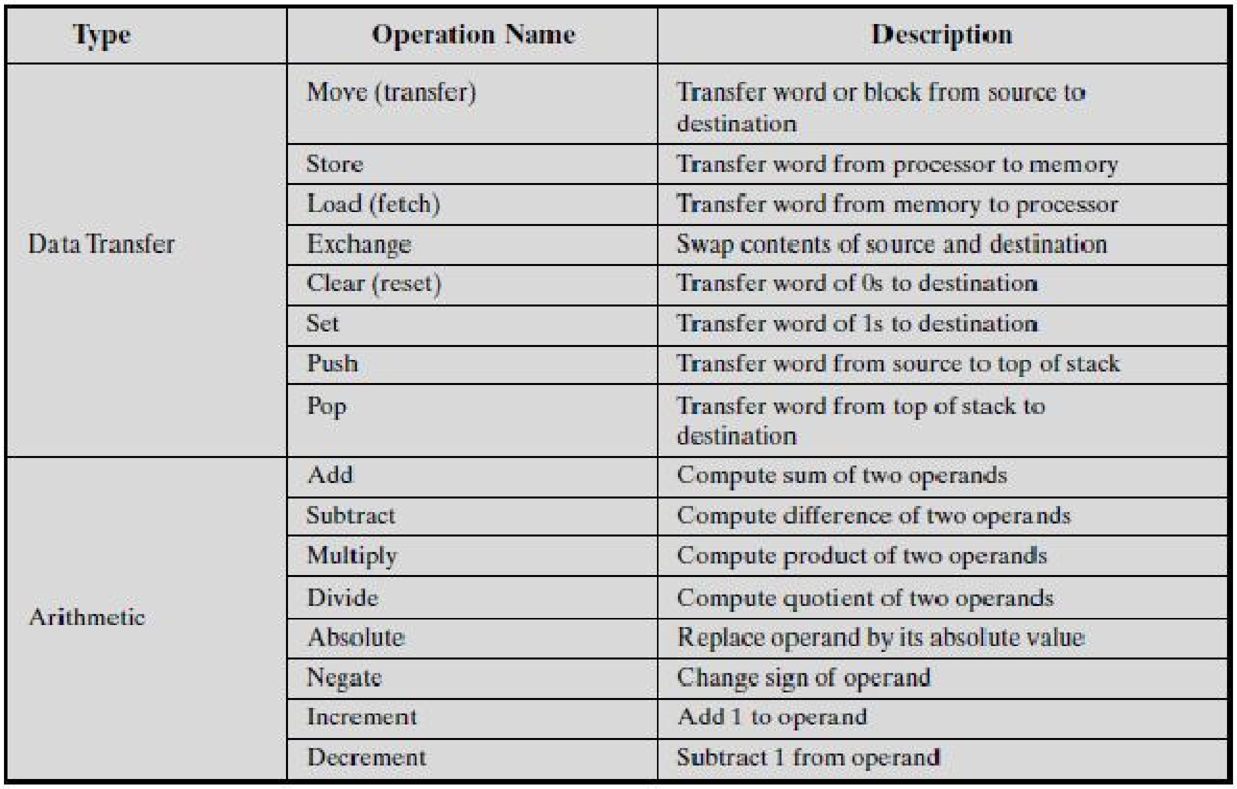
Hard Disk, Floppy Disk, Magenetic Tape	------ These are magnetic devices,
CD-ROM	------ is optical device
Thumb drive (or pen drive)	------ is semiconductor memory.

Machine instructions and programs

Machine Instructions are **commands or programs** written in machine code of a machine (computer) that it can recognize and execute. The number of different opcodes varies widely from machine to machine. A computer must have instructions capable of performing four types of operations [1]

- Data transfer
- Arithmetic
- Logical
- I/O
- Transfer of control

Common Instruction Set Operations



Type	Operation Name	Description
Data Transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand

Type	Operation Name	Description
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end
Transfer of Control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued
Input/Output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination

Register transfer notation

Register is a very fast computer memory, used to store data/instruction in-execution.

A **Register** is a group of flip-flops with each flip-flop capable of storing **one bit** of information. An *n-bit* register has a group of *n flip-flops* and is capable of storing binary information of *n-bits*.**‹**

The contents of a location are denoted by placing square brackets around the name of the location ‹ For example, **R1← [LOC]** means that the contents of memory location LOC are transferred into processor register R1

As another example, **R3 ← [R1]+[R2]** means that adds the contents of registers R1 and R2, and then places their sum into register R3

Symbolic Designation	**Description**
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)'	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.
R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).
R1 ← R1 + 1	Increment the contents of R1 by 1.
R1 ← R1 – 1	Decrement the contents of R1 by 1.

Types of instructions

- Zero-address instruction
- One-address instruction
- Two-address instruction
- Three-address instruction

Zero-address instruction

For example, **PUSH**: store operands in a structure called a pushdown stack

One-address instruction

Instruction form: Operation Destination

For example, **Add A**: add the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator

As another example, **Load A**: copies the contents of memory location A into the accumulator

Two-address instruction

Instruction form: Operation Source, Destination

For example, **Add A, B**: performs the operation **B ← [A]+[B].** When the sum is calculated, the result is sent to the memory and stored in location B

As another example, **Move B, C**: performs the operation **C ← [B],** leaving the contents of location B unchanged

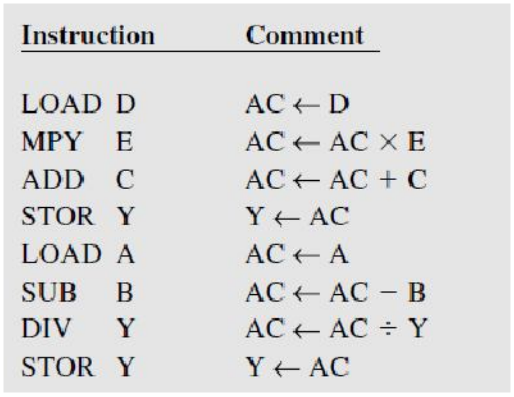
Three-address instruction

Instruction form: Operation Source1, Source2, Destination

For example, **Add A, B, C**: adds A and B, and the result is sent to the memory and stored in location C

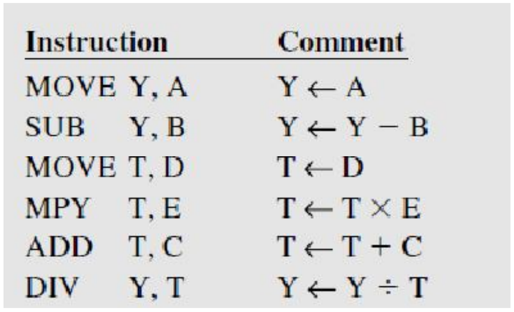
Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

One-address instructions:



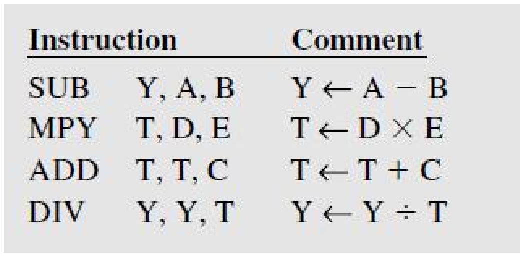
Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

Two-address instructions:



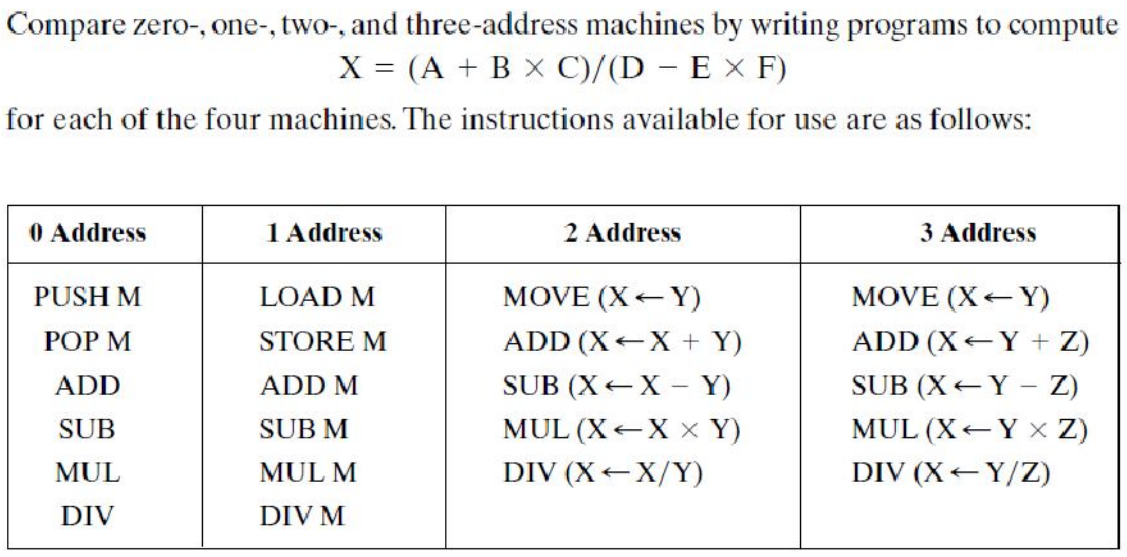
Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

Three-address instructions:



Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

# Exercise



Compare zero-, one-, two-, and three-address machines by writing programs to compute

$$X = (A + B \times C)/(D - E \times F)$$

for each of the four machines. The instructions available for use are as follows:

0 Address	1 Address	2 Address	3 Address
PUSH M	LOAD M	MOVE $(X \leftarrow Y)$	MOVE $(X \leftarrow Y)$
POP M	STORE M	ADD $(X \leftarrow X + Y)$	ADD $(X \leftarrow Y + Z)$
ADD	ADD M	SUB $(X \leftarrow X - Y)$	SUB $(X \leftarrow Y - Z)$
SUB	SUB M	MUL $(X \leftarrow X \times Y)$	MUL $(X \leftarrow Y \times Z)$
MUL	MUL M	DIV $(X \leftarrow X/Y)$	DIV $(X \leftarrow Y/Z)$
DIV	DIV M		

Instruction sets: Instruction formats

Types of Addressing Modes

Below we have discussed different types of addressing modes one by one [1]:

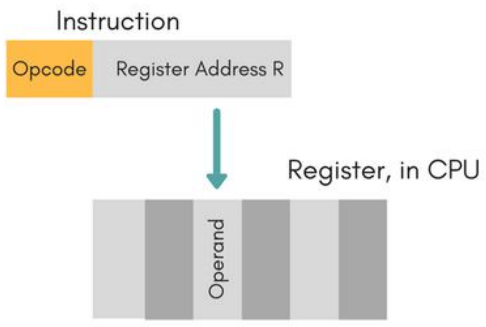
Immediate Mode

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.

Register Mode

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.



Advantages

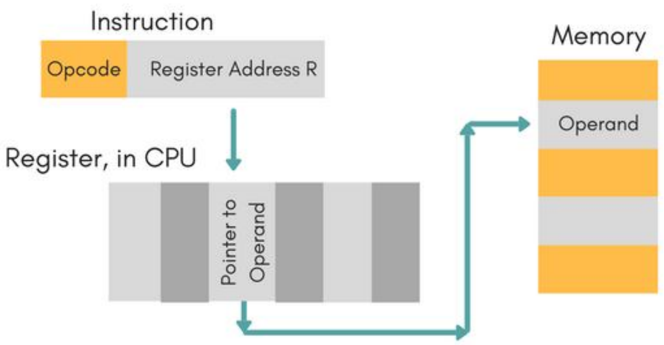
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

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

Register Indirect Mode

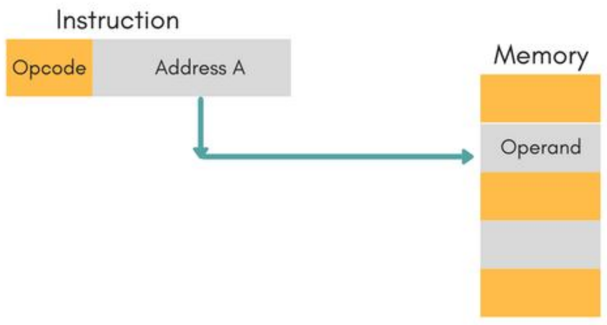
In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory. Thus, the register contains the address of operand rather than the operand itself.



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.

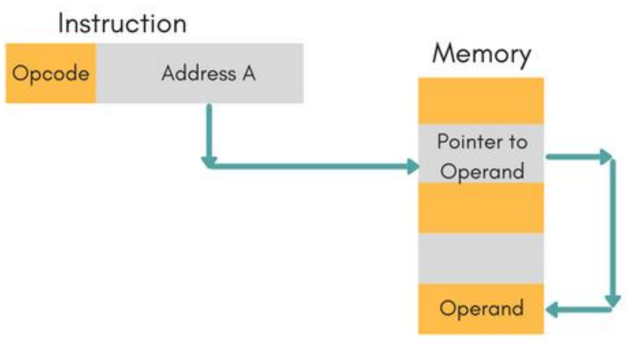


For Example: ADD R1, 4000 - In this the 4000 is effective address of operand.

NOTE: Effective Address is the location where operand is present.

Indirect Addressing Mode

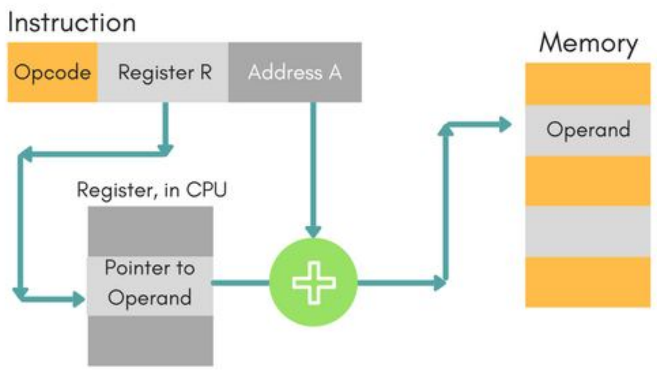
In this, the address field of instruction gives the address where the effective address is stored in memory. This slows down the execution, as this includes multiple memory lookups to find the operand.



Displacement Addressing Mode

In this the contents of the indexed register is 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.



(a) Relative Addressing Mode

It is a version of Displacement addressing mode.

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

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)

(b) 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.

Stack Addressing Mode

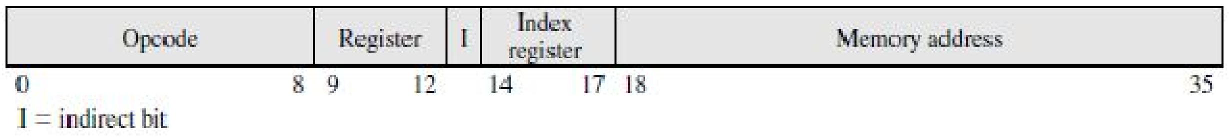
In this mode, operand is at the top of the stack. For example: ADD, this instruction will *POP* top two items from the stack, add them, and will then *PUSH* the result to the top of the stack.

Instruction Format

- Instruction format defines the layout of the bits of an instruction, in terms of its component fields [1].
- An instruction format must include an opcode and implicitly or explicitly, zero or more operands.
- Each explicit operand is referenced using one of the addressing modes.
- Instruction format is affected by, memory size, memory organization, bus structure, processor complexity, and processor speed.

For example: The PDP-10 has a 36-bit word length and a 36-bit instruction length. The fixed instruction format is shown in figure below.

- The opcode occupies 9 bits, allowing up to 512 operations (i.e. 2^9). In fact, a total of 365 different instructions are defined.
- Register occupies 4 bits, one of which is one of 16 general-purpose registers.
- Index register occupies 4 bits, one of which is one of 16 addressing modes
- The other operand reference starts with an 18-bit memory address field. This can be used as an immediate operand or a memory address (2^{18}). In the latter usage, both indexing and indirect addressing are allowed.
- The same general-purpose registers are also used as index registers.



Example: Design a variable-length opcode to allow all of the following to be encoded in a 36-bit instruction:

a) Instructions with two 15-bit addresses and one 3-bit register number
b) Instructions with one 15-bit address and one 3-bit register number

Performance issues software

- INSTRUCTION EXECUTION RATE A processor is driven by a clock with a constant frequency f or, equivalently, a constant cycle time t, where t=1/f .
- Define the instruction count, I_c, for a program as the number of machine instructions executed for that program until it runs to completion. Note that this is the number of instruction executions, not the number of instructions in the object code of the program.
- An important parameter is the average cycles per instruction CPI for a program. If all instructions required the same number of clock cycles, then CPI would be a constant value for a processor. However, on any give processor, the number of clock cycles required varies for different types of instructions, such as load, store, branch, and so on.
- Let CPI_i be the number of cycles required for instruction type i. and
- I_i be the number of executed instructions of type i for a given program.

Then we can calculate an overall CPI as follows[1]:

$$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times I_i)}{I_c}$$

The processor time T needed to execute a given program can be expressed as

$$T = I_c * CPI * t$$

A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (MIPS), referred to as the **MIPS rate**.
We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

For example, consider the execution of a program which results in the execution of 2 million instructions on a 400-MHz processor. The program consists of four major types of instructions. The instruction mix and the CPI for each instruction type are given below based on the result of a program trace experiment:

Instruction Type	CPI	Instruction Mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

The average CPI when the program is executed on a uniprocessor with the above trace results is

$$CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24.$$

The corresponding MIPS rate is

$$(400 \times 10^6)/(2.24 \times 10^6) \approx 178.$$

Assembly Language

- A complete set of symbolic names and rules for their use constitute a programming language, generally referred to as an assembly language.
- Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an assembler.
- When the assembler program is executed, it reads the user program, analyzes it, and then generates the desired machine language program.
- The user program in its original alphanumeric text format is called a source program, and the assembled machine language program is called an object program.

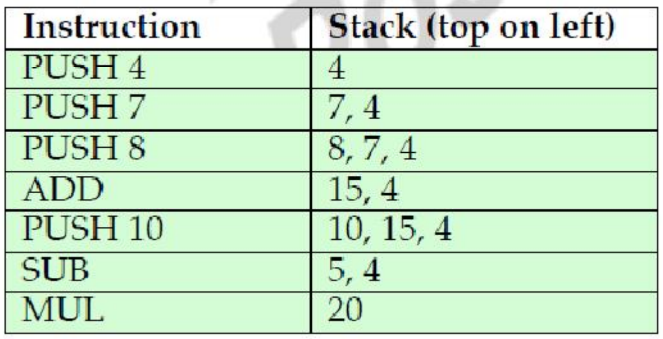
Stack and Queue

- A stack is a list of data elements, usually words or bytes, with the accessing restriction that elements can be added or removed at one end of the list only [5]
    - It is also called a last-in-first-out (LIFO) stack
    - A stack has two basic operations: push and pop
    - The terms push and pop are used to describe placing a new item on the stack and removing the top item from the stack, respectively.
- Another useful data structure that is similar to the stack is called a queue
    - Data are stored in and retrieved from a queue on a first-in-firstout (FIFO) basis
    - Two pointers are needed to keep track of the two ends of the queue

Example: Assume a stack-oriented processor that includes the stack operations PUSH and POP. Arithmetic operations automatically involve the top one or two stack elements. Begin with an empty stack. What stack elements remain after the following instructions are executed?

```
PUSH 4
PUSH 7
PUSH 8
ADD
PUSH 10
SUB
MUL
```

Answer:



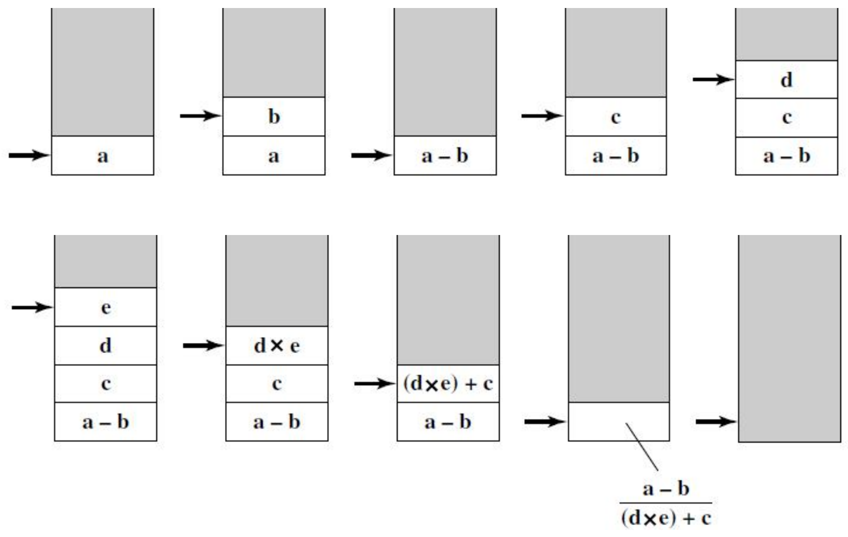
Instruction	Stack (top on left)
PUSH 4	4
PUSH 7	7, 4
PUSH 8	8, 7, 4
ADD	15, 4
PUSH 10	10, 15, 4
SUB	5, 4
MUL	20

Expression Evaluation[1]

- Mathematical formulas are usually expressed in **infix** notation. In this form, a binary operator appears between the operands (e.g., A + B). Generally, multiplication takes precedence over addition, so that a + b x c is equivalent a + (b x c).
- An alternative technique is known as **reverse Polish**, or **postfix**, notation. In this notation, the operator follows its two operands. For example,

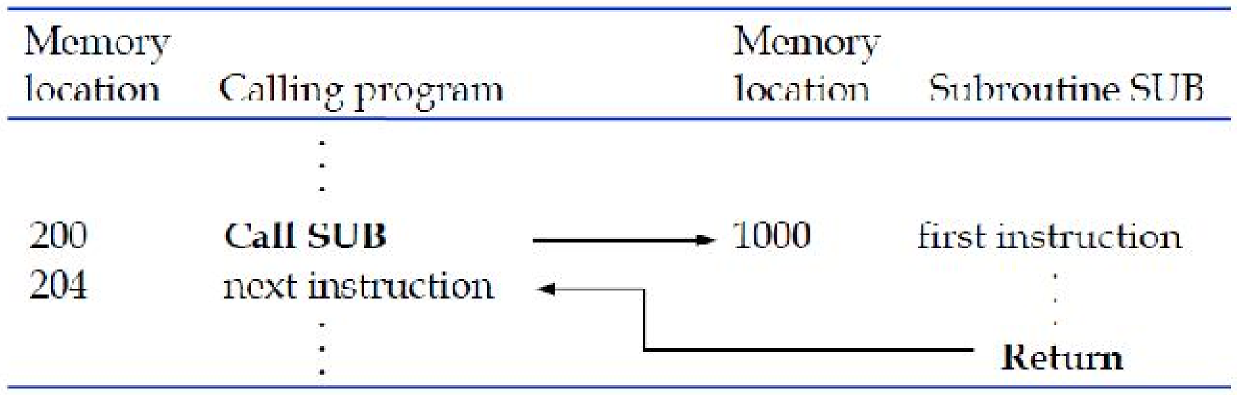
$a + b$	becomes $a\ b\ +$
$a + (b \times c)$	becomes $a\ b\ c \times +$
$(a + b) \times c$	becomes $a\ b\ +\ c \times$

- Note that, no parentheses are required when using **reverse Polish**. The advantage of **postfix** notation is that an expression in this form is easily evaluated using a **stack**.
- An expression in **postfix** notation is scanned from **left to right**. For each element of the expression, the following rules are applied:
- **1.** If the element is a variable or constant, push it onto the stack.
- **2.** If the element is an operator, pop the top two items of the stack, perform the operation, and push the result.
- After the entire expression has been scanned, the result is on the top of the stack.
- **Example:** Use of Stack to Compute f = (a - b) / [(d * e) + c]



Subroutine

- In a given program, it is often necessary to perform a particular subtask many times on different data values. Such a subtask is called a subroutine[5].

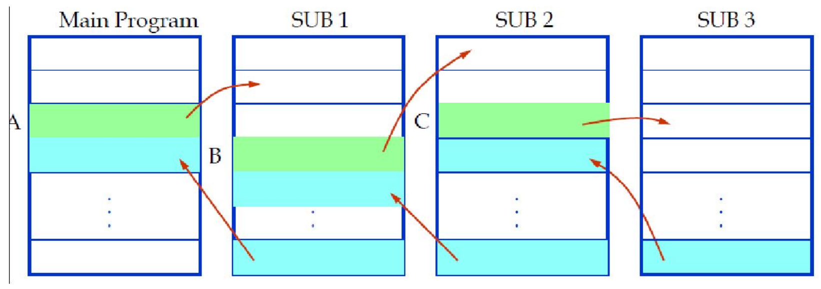


- The location where the calling program resumes execution is the location pointed by the updated PC while the Call instruction is being executed. Hence the contents of the PC must be saved by the Call instruction to enable correct return to the calling program

Subroutine Nesting

- A common programming practice, called subroutine nesting, is to have one subroutine call another.
- Subroutine nesting call be carried out to any depth. Eventually, the last subroutine called completes its computations and returns to the subroutine that called it.
- The return address needed for this first returns is the last one generated in the nested call sequence. That is, return addresses are generated and used in a last-in-first-out order.
- Many processors do this by using a stack pointer and the stack pointer points to a stack called the processor stack.

Example of Subroutine Nesting



References:

1. Computer Organization and Architecture: Designing for Performance, 8th Edition, Authors: William Stallings Publisher: Prentice-Hall India.
2. https://nptel.ac.in/courses/106/103/106103068/
3. https://en.wikipedia.org/wiki/Von_Neumann_architecture
4. https://en.wikipedia.org/wiki/Harvard_architecture
5. http://www.ee.ncu.edu.tw/~jfli/computer/lecture/ch03.pdf

-------------------------------End of Unit-1----------------------------------