THANKS TO SECTION-11 & SECTION 24

LECTURE NOTES

Course Title	Computer Organization and Architecture			Course Type			Theory		
Course Code	E2UC505T			Class			5th Semester B.Tech. Core and All specialization		
Instruction delivery	Activity	Credits	Credit Hours	Total Number of Classes per Semester				Assessment in Weightage	
				Theory	Tutorial	Practical	Self-Learning	CIE	SEE
	Lecture	3	3						
	Tutorial	1	1						
	Practical	0	0						
	Self-Learning	0	6						
	Total	4	10	45	15	0	90	**50%**	**50%**

Syllabus

Unit 1 Introduction: Functional units of digital system and their interconnections, buses, bus architecture, types of buses and bus arbitration. Register, bus and memory transfer. Processor organization, general registers organization, stack organization and addressing modes

Unit 2 Arithmetic and logic unit: Look ahead carries adders. Multiplication: Signed operand multiplication, Booths algorithm and array multiplier. Division and logic operations. Floating point arithmetic operation, Arithmetic & logic unit design. IEEE Standard for Floating Point Numbers

Unit 3 Control Unit: Instruction types, formats, instruction cycles and sub cycles (fetch and execute etc), micro operations, execution of a complete instruction.

MID-01

Program Control, Reduced InstructionSet Computer, Pipelining. Hardwire and micro programmed control: micro programme sequencing,concept of horizontal and vertical microprogramming.

Unit 4 Memory: Basic concept and hierarchy, semiconductor RAM memories, 2D & 2 1/2D memory organization. ROM memories. Cache memories: concept and design issues & performance, address mapping and replacement Auxiliary memories: magnetic disk, magnetic tape and optical disks Virtual memory: concept implementation.

Unit 5 Input / Output: Peripheral devices, I/O interface, I/O ports, Interrupts: interrupt hardware, types of interrupts and exceptions. Modes of Data Transfer: Programmed I/O, interrupt initiated I/O and Direct Memory Access., I/O channels and processors. Serial Communication: Synchronous & asynchronous communication, standard communication interfaces

THANKS TO SECTION-11 & SECTION 24

Index:

1. Introduction 5-24

1.1. Functional Units of Digital System And Their Interconnections 5

1.2. Buses & Bus Architecture, ... 7

1.3. Types of Buses 8

1.4. Bus Arbitration 9

1.5. Register 13

1.6. Bus and Memory Transfer. 14

1.7. Processor Organization. 18

1.7.1. General Registers Organization. 19

1.7.2. Stack Organization 21

1.8. Addressing Modes 21

2. Arithmetic And Logic Unit 25-42

2.1. Look Ahead Carries Adders 25

2.2. Multiplication 28

2.2.1. Signed Operand Multiplication 28

2.2.2. Booths Algorithm And Array Multiplier 30

2.3. Division And Logic Operations 33

2.4. Floating Point Arithmetic Operation 35

2.5. IEEE Standard For Floating Point Numbers 38

2.6. Arithmetic & Logic Unit Design 40

3. Control Unit 43-50

3.1. Instruction Types 43

3.2. Formats 43

3.3. Instruction Cycles And Sub Cycles (Fetch And Execute Etc) 46

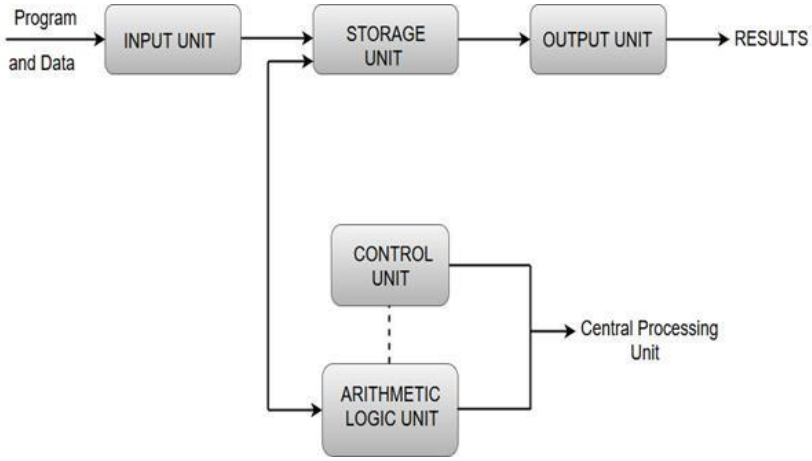
3.4. Micro Operations 48

3.5. Execution of A Complete Instruction 49

UNIT-1

1.1. Functional Units of Digital System

- A computer organization describes the functions and design of the various units of a digital system.
- A general-purpose computer system is the best-known example of a digital system. Other examples include telephone switching exchanges, digital voltmeters, digital counters, electronic calculators and digital displays.
- Computer architecture deals with the specification of the instruction set and the hardware units that implement the instructions.
- Computer hardware consists of electronic circuits, displays, magnetic and optic storage media and also the communication facilities.
- Functional units are a part of a CPU that performs the operations and calculations called for by the computer program.
- Functional units of a computer system are parts of the CPU (Central Processing Unit) that performs the operations and calculations called for by the computer program. A computer consists of five main components namely, Input unit, Central Processing Unit, Memory unit Arithmetic & logical unit, Control unit and an Output unit.



Input unit

- Input units are used by the computer to read the data. The most commonly used input devices are keyboards, mouse, joysticks, trackballs, microphones, etc.
- However, the most well-known input device is a keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Central processing unit: Central processing unit commonly known as CPU can be referred as an electronic circuitry within a computer that carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Memory unit

- The Memory unit can be referred to as the storage area in which programs are kept which are running, and that contains data needed by the running programs.
- The Memory unit can be categorized in two ways namely, primary memory and secondary memory.
- It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.
- Primary storage is the fastest memory that operates at electronic speeds. Primary memory contains a large number of semiconductor storage cells, capable of storing a bit of information. The word length of a computer is between 16-64 bits.
- It is also known as the volatile form of memory, means when the computer is shut down, anything contained in RAM is lost.
- Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor.
- The most common examples of primary memory are RAM and ROM.
- Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.
- It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.
- The most common examples of secondary memory are magnetic disks, magnetic tapes, and optical disks.

Arithmetic & logical unit: Most of all the arithmetic and logical operations of a computer are executed in the ALU (Arithmetic and Logical Unit) of the processor. It performs arithmetic operations like addition, subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

Control unit

The control unit is a component of a computer's central processing unit that coordinates the operation of the processor. It tells the computer's memory, arithmetic/logic unit and input andoutput devices how to respond to a program's instructions.

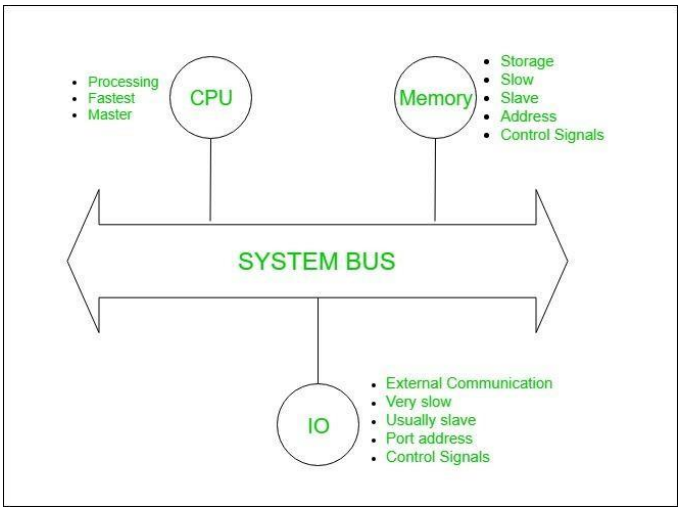
- The control unit is also known as the nerve center of a computer system.
- Let's us consider an example of addition of two operands by the instruction given as Add LOCA, RO. This instruction adds the memory location LOCA to the operand in the register RO and places the sum in the register RO. This instruction internally performs several steps.

Output Unit

- The primary function of the output unit is to send the processed results to the user. Output devices display information in a way that the user can understand.
- Output devices are pieces of equipment that are used to generate information or any other response processed by the computer. These devices display information that has been held or generated within a computer.
- The most common example of an output device is a monitor.

1.2. Buses, Bus Architecture

A bus that connects major components (CPU, memory and I/O devices) of a computer system is called as a System Bus. A bus is a set of electrical wires (lines) that connects the various hardware components of a computer system. It works as a communication pathway through which information flows from one hardware component to the other hardware component. Bus Architecture is shown in figure below.



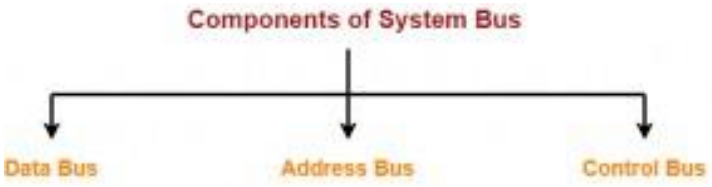
- A computer system is made of different components such as memory, ALU, registers etc.
- Each component should be able to communicate with other for proper execution of instructions and information flow.
- If we try to implement a mesh topology among different components, it would be really expensive.

- So, we use a common component to connect each necessary component i.e. BUS.

1.3. Types of buses

System bus contains 3 categories of lines used to provide the communication between the CPU, memory and IO named as:

1. Data Bus
2. Address Bus
3. Control Bus



Data Bus: As the name suggests, data bus is used for transmitting the data / instruction from CPU to memory/IO and vice-versa. It is bi-directional. The width of a data bus refers to the number of bits (electrical wires) that the bus can carry at a time. Each line carries 1 bit at a time. So, the number of lines in data bus determine how many bits can be transferred parallelly. The width of data bus is an important parameter because it determines how much data can be transmitted at one time. The wider the bus width, faster would be the data flow on the data bus and thus better would be the system performance.

Examples-

- A 32-bit bus has thirty two (32) wires and thus can transmit 32 bits of data at a time.
- A 64-bit bus has sixty four (64) wires and thus can transmit 64 bits of data at a time.

Address Bus : As the name suggests, address bus is used to carry address from CPU to memory/IO devices. It is used to identify the particular location in memory. It carries the source or destination address of data i.e. where to store or from where to retrieve the data. It is uni-directional.

Example- When CPU wants to read or write data, it sends the memory read or memory write control signal on the control bus to perform the memory read or write operation from the main memory and the address of the memory location is sent on the address bus.

- If CPU wants to read data stored at the memory location (address) 4, the CPU send the value 4 in binary on the address bus.

The width of address bus determines the amount of physical memory addressable by the processor.

- In other words, it determines the size of the memory that the computer can use.
- The wider is the address bus, the more memory a computer will be able to use.
- The addressing capacity of the system can be increased by adding more address lines.

Examples-

- An address bus that consists of 16 wires can convey 2^{16} (= 64K) different addresses.
- An address bus that consists of 32 wires can convey 2^{32} (= 4G) different addresses.

Control Bus : As the name suggests, control bus is used to transfer the control and timing signals from one component to the other component. The CPU uses control bus to communicate with the devices that are connected to the computer system. The CPU transmits different types of control signals to the system components. It is bi-directional.

Typical control signals hold by control bus

Memory read – Data from memory address location to be placed on data bus.

Memory write – Data from data bus to be placed on memory address location.

I/O Read – Data from I/O address location to be placed on data bus.

I/O Write – Data from data bus to be placed on I/O address location.

Other control signals hold by control bus are interrupt, interrupt acknowledge, bus request, bus grant and several others. The type of action taking place on the system bus is indicated by these control signals.

Example-

When CPU wants to read or write data, it sends the memory read or memory write control signal on the control bus to perform the memory read or write operation from the main memory. Similarly, when the processor wants to read from an I/O device, it generates the I/O read signal.

1.4. System Bus Arbitration

Bus Arbitration is the procedure by which the active bus master accesses the bus, relinquishes control of it, and then transfers it to a different bus-seeking processor unit. A bus master is a controller that can access the bus for a given instance.

A conflict could occur if multiple DMA controllers, other controllers, or processors attempt to access the common bus simultaneously, yet only one is permitted to access. Bus master status can

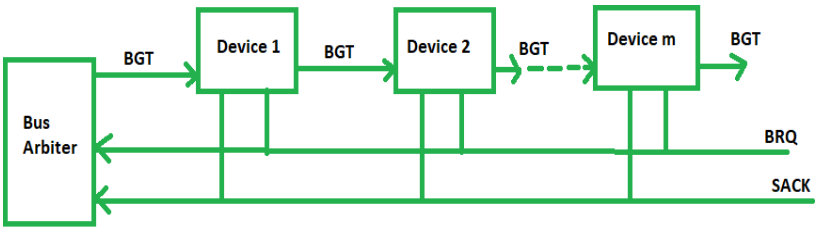
only be held by one processor or controller at once. By coordinating the actions of all devices seeking memory transfers, the Bus Arbitration method is used to resolve these disputes

There are two approaches to bus arbitration:

- **Centralized Bus Arbitration** - In which the necessary arbitration is carried out by a lone bus arbitrator.
- **Distributive Bus Arbitration** - In which every device takes part in choosing the new bus master. A 4bit identification number is allocated to each device on the bus. The created ID will decide the device's priority

a) **Centralized Bus Arbitration Methodologies :** There are three methods of Centralized Bus Arbitration, which are listed below:

i. **Daisy Chaining method:** It is a simple and cheaper method where all the bus masters use the same line for making bus requests. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, therefore any other requesting module will not receive the grant signal and hence cannot access the bus. During any bus cycle, the bus master may be any device – the processor or any DMA controller unit, connected to the bus.

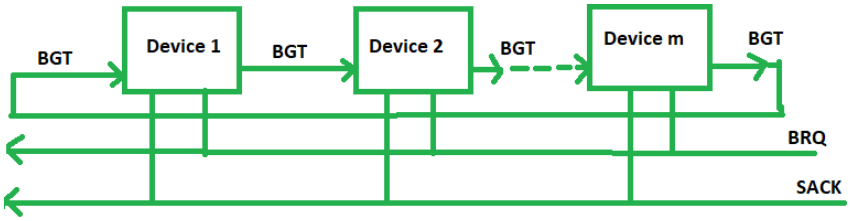


Daisy chained bus arbitration

Advantages:

- Simplicity and Scalability.
  - The value of priority assigned to a device depends on the position of the master bus.
  - Propagation delay arises in this method.
  - If one device fails then the entire system will stop working.

ii. **Polling or Rotating Priority method:** In this, the controller is used to generate the address for the master(unique priority), the number of address lines required depends on the number of masters connected in the system. The controller generates a sequence of master addresses. When the requesting master recognizes its address, it activates the busy line and begins to use the bus.



Rotating priority bus arbitration

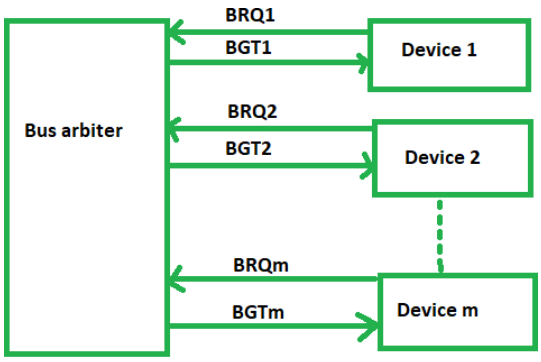
Advantages –

- This method does not favor any particular device and processor.
- The method is also quite simple.

Disadvantages –

- Adding bus masters is difficult as increases the number of address lines of the circuit.
- If one device fails then the entire system will not stop working.

iii. **Fixed priority or Independent Request method:** In this, each master has a separate pair of bus request and bus grant lines and each pair has a priority assigned to it. The built-in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal.



Fixed priority bus arbitration method

Advantages –

This method generates a fast response.

Disadvantages –

Hardware cost is high as a large no. of control lines is required.

Distributed BUS Arbitration:

In distributed arbitration, all devices participate in the selection of the next bus master.

- In this scheme each device on the bus is assigned a4-bit identification number.
- The number of devices connected on the bus when one or more devices request for the control of bus, they assert the start-arbitration signal and place their 4-bit ID numbers on arbitration lines, ARB0 through ARB3.
- These four arbitration lines are all open-collector. Therefore, more than one device can place their 4-bit ID number to indicate that they need to control of bus. If one device puts 1 on the bus line and another device puts 0 on the same bus line, the bus line status will be 0. Device reads the status of all lines through inverters buffers so device reads bus status 0as logic 1. Scheme the device having highest ID number has highest priority.
- When two or more devices place their ID number on bus lines then it is necessary to identify the highest ID number on bus lines then it is necessary to identify the highest ID number from the status of bus line. Consider that two devices A and B, having ID number 1 and 6, respectively are requesting the use of the bus.
- Device A puts the bit pattern 0001, and device B puts the bit pattern 0110. With this combination the status of bus-line will be 1000; however because of inverter buffers code seen by both devices is 0111.

- Each device compares the code formed on the arbitration line to its own ID, starting from the most significant bit. If it finds the difference at any bit position, it disables its drives at that bit position and for all lower-order bits.
- It does so by placing a 0 at the input of their drive. In our example, device detects a different on line ARB2 and hence it disables its drives on line ARB2, ARB1 and ARB0. This causes the code on the arbitration lines to change to 0110. This means that device B has won the race.
- The decentralized arbitration offers high reliability because operation of the bus is not dependent on any single device.

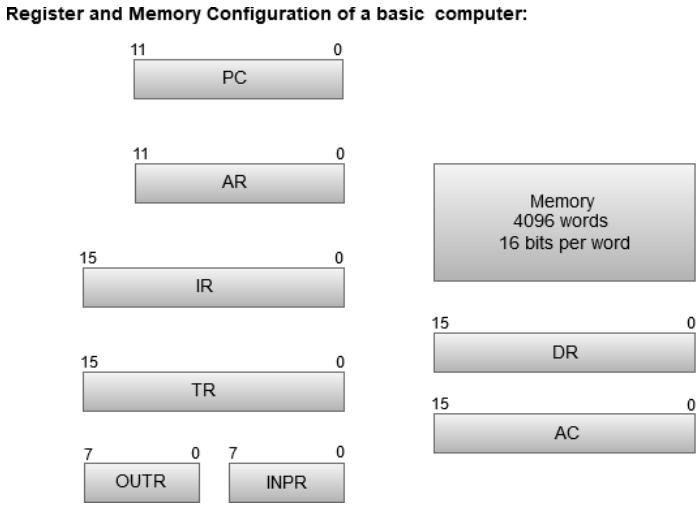
1.5. Register

- Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers.
- A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).
- The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to calculate the address of the next instruction after the execution of the current instruction is completed.

Following is the list of some of the most common registers used in a basic computer:

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

The following image shows the register and memory configuration for a basic computer.



- The Memory unit has a capacity of 4096 words, and each word contains 16 bits.
- The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
- The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
- The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
- The Accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the Instruction register (IR).
- The Temporary Register (TR) is used for holding the temporary data during the processing.
- The Input Registers (IR) holds the input characters given by the user.
- The Output Registers (OR) holds the output after processing the input data.

1.6. Bus and Memory Transfers

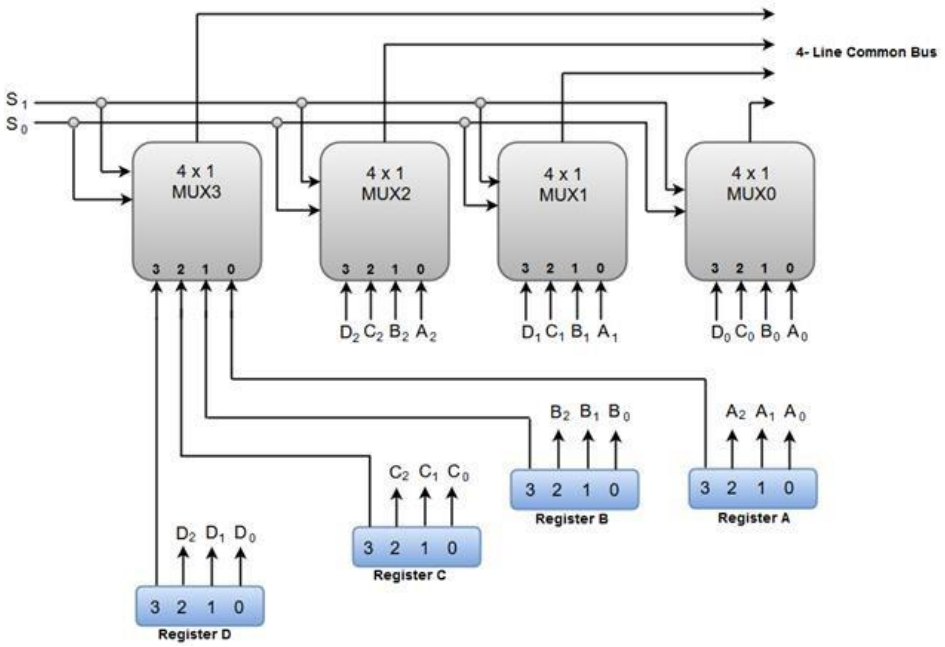
A digital system composed of many registers, and paths must be provided to transfer information from one register to another. The number of wires connecting all of the registers will be excessive if separate lines are used between each register and all other registers in the system.

A bus structure, on the other hand, is more efficient for transferring information between registers in a multi-register configuration system.

A bus consists of a set of common lines, one for each bit of register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during a particular register transfer.

The following block diagram shows a Bus system for four registers. It is constructed with the help of four 4 * 1 Multiplexers each having four data inputs (0 through 3) and two selection inputs (S1 and S2).

Bus System for 4 Registers:



The two selection lines S1 and S2 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus.

When both of the select lines are at low logic, i.e. S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that forms the bus. This, in turn, causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

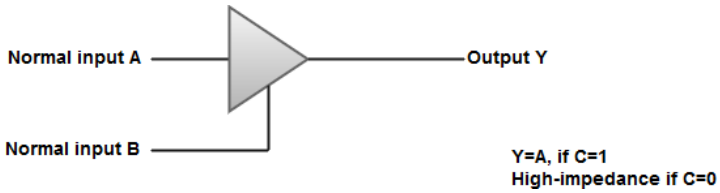
Similarly, when S1S0 = 01, register B is selected, and the bus lines will receive the content provided by register B.

The following function table shows the register that is selected by the bus for each of the four possible binary values of the Selection lines.

S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

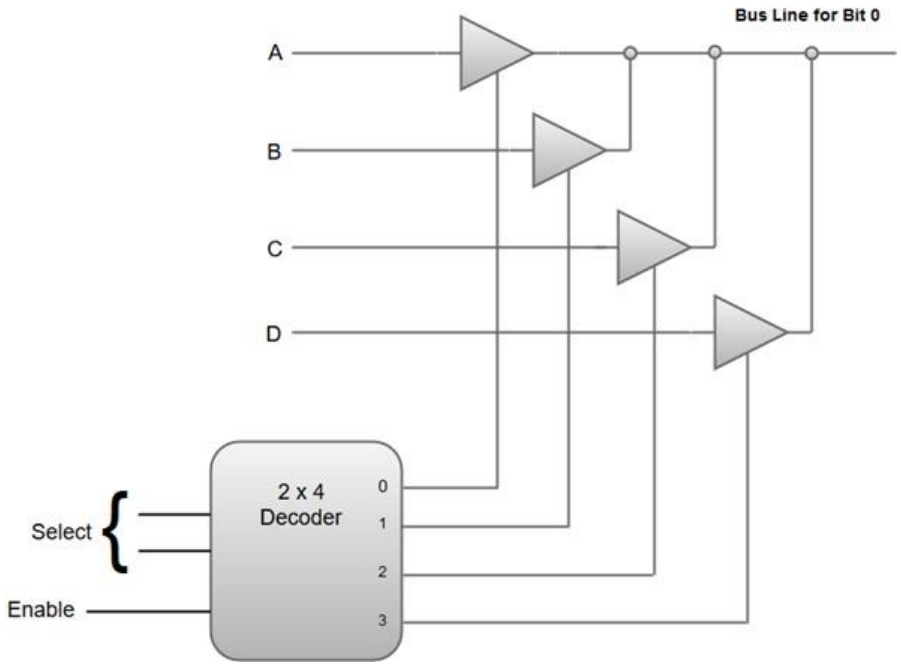
A bus system can also be constructed using **three-state gates** instead of multiplexers. The **three state gates** can be considered as a digital circuit that has three gates, two of which are signals equivalent to logic 1 and 0 as in a conventional gate. However, the third gate exhibits a high-impedance state. The most commonly used three state gates in case of the bus system is a **buffer gate**.

The graphical symbol of a three-state buffer gate can be represented as:



The following diagram demonstrates the construction of a bus system with three-state buffers.

Bus line with three state buffer:



- The outputs generated by the four buffers are connected to form a single bus line.
- Only one buffer can be in active state at a given point of time.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- A 2 * 4 decoder ensures that no more than one control input is active at any given point of time.

Memory Transfer

Most of the standard notations used for specifying operations on memory transfer are stated below.

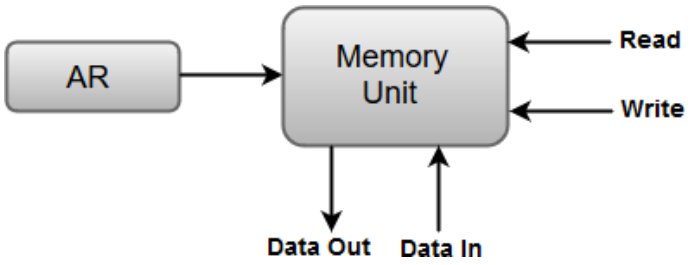
- The transfer of information from a memory unit to the user end is called a **Read** operation.
- The transfer of new information to be stored in the memory is called a **Write** operation.
- A memory word is designated by the letter **M**.
- We must specify the address of memory word while writing the memory transfer operations.
- The **address register** is designated by **AR** and the **data register** by **DR**.
- Thus, a read operation can be stated as:

Read: DR ← M [AR]

- The **Read** statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR).
- And the corresponding write operation can be stated as:

Write: M [AR] ← R1

- The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).



1.7. Processor organizations

Processor Organizations:

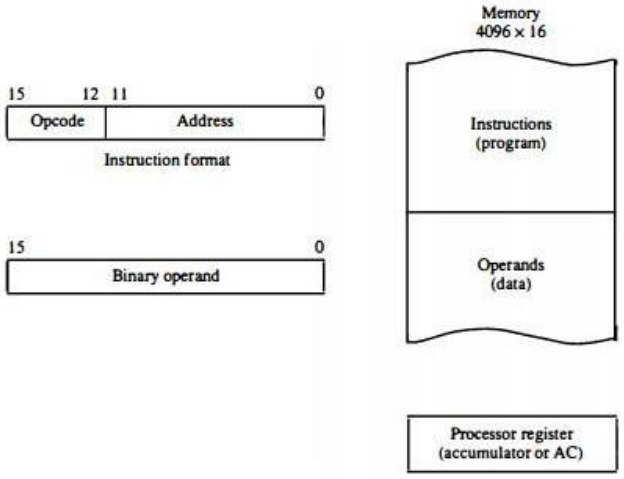
There are 3 types of Processor organizations

1. Stored Program Organization
2. General Register Organization
3. Stack Organization

1. Stored Program Organization:

The simplest way to organize a computer is to have one processor register and instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address. The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register. Figure 5.1 depicts this type of organization. Instructions are stored in one section of memory and data in another. For a memory unit with 4096 words we need 12 bits to specify an address since 212 = 4096. If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated op code) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand. The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory. It then executes the operation specified by the operation code.

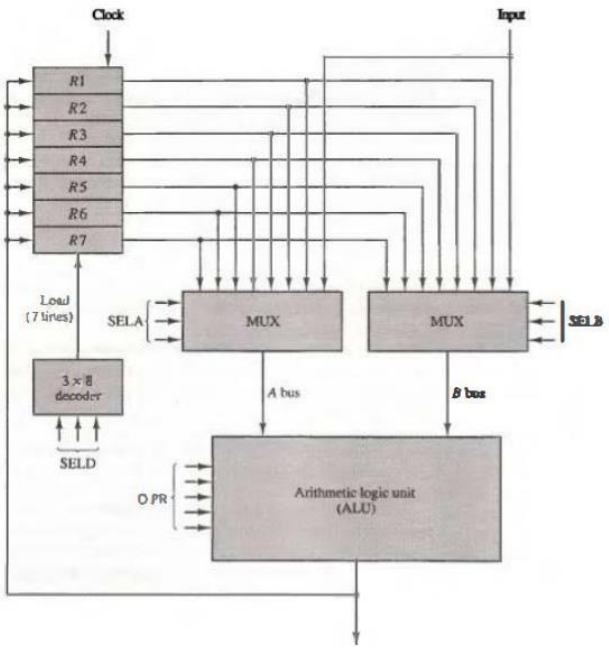
Figure 5-1 Stored program organization.



Computers that have a single-processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and the content of AC.
Computers that have a single-processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and the content of AC.

2. General Register Organization

- Memory locations are needed for storing pointers, counters, return addresses, temporary results, and partial products during multiplication.
- Having to refer to memory locations for such applications is time consuming because memory access is the most time-consuming, operation in a computer.
- It is more convenient and more efficient to store these intermediate values in processor registers.
- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various microoperations.
- Hence it is necessary to provide a common unit that can perform all the arithmetic, logic, and shift microoperations in the processor.
- A bus organization for seven CPU registers is shown in Fig. 2. The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection lines in each multiplexer select one register or the input data for the particular bus.
- The A and B buses form the inputs to a common arithmetic logic unit (ALU).
- The operation selected in the ALU determines the arithmetic or logic micro-operation that is to be performed.
- The result of the microoperation is available for output data and also goes into the inputs of all the registers.
- The register that receives the information from the output bus is selected by a decoder.
- The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.



(a) Block diagram

(b) Control word

Figure 2 Register set with common ALU.

- the control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system. For example, to perform the operation R1 ← R2 + R3
- the control must provide binary selection variables to the following selector inputs:
- MUX A selector (SELA): to place the content of R2 into bus A.
- MUX B selector (SELB): to place the content o f R 3 into bus B.
- ALU operation selector (OPR): to provide the arithmetic addition A + B.
- Decoder destination selector (SELD): to transfer the content of the output bus into R1.
- The four control selection variables are generated in the control unit and must be available at the beginning of a clock cycle.
- The data from the two source registers propagate through the gates in the multiplexers and the ALU, to the output bus, and into the inputs of the destination register, all during the clock cycle interval.
- Then, when the next clock transition occurs, the binary information from the output bus is transferred into R1.

- To achieve a fast response time, the ALU is constructed with high-speed circuits.
-

3. Stack Organization

Stack is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The stack in digital computers is a group of memory locations with a register that holds the address of top of element. This register that holds the address of top of element of the stack is called Stack Pointer.

The two operations of a stack are:

1. Push: Inserts an item on top of stack.
2. Pop: Deletes an item from top of stack.

Implementation of Stack

1. Register Stack
2. Memory Stack.

Register Stack

A stack can be organized as a collection of finite number of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top of element of the stack.

Memory Stack

A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as stack pointer.

1.8. Addressing Modes

The addressing modes help us specify the way in which an operand's effective address is represented in any given instruction. Some addressing modes allow referring to a large range of areas efficiently, like some linear array of addresses along with a list of addresses. The addressing modes describe an efficient and flexible way to define complex effective addresses.

The programs are generally written in high-level languages, as it's a convenient way in which one can define the variables along with the operations that a programmer performs on the variables. This program is later compiled so as to generate the actual machine code. A machine code includes low-level instructions.

A set of low-level instructions has operands and opcodes. An addressing mode has no relation with the opcode part. It basically focuses on presenting the address of the operand in the instructions.

Addressing Modes Types

The addressing modes refer to how someone can address any given memory location. Five different addressing modes or five ways exist using which this can be done.

You can find the list below, showing the various kind of addressing modes:

- Implied Mode
- Immediate Mode
- Register Mode
- Register Indirect Mode
- Autodecrement Mode
- Autoincrement Mode
- Direct Address Mode
- Indirect Address Mode
- Indexed Addressing Mode

Before getting into discussing the addressing modes, one must understand more about the "effective address" term.

Effective Address (EA)

The effective address refers to the address of an exact memory location in which an operand's value is actually present. Let us now explain all of the addressing modes.

Implied Mode

In the implied mode, the operands are implicitly specified in the definition of instruction. For instance, the "complement accumulator" instruction refers to an implied-mode instruction. It is because, in the definition of the instruction, the operand is implied in the accumulator register. All the register reference instructions are implied-mode instructions that use an accumulator.

Immediate Mode

In the immediate mode, we specify the operand in the instruction itself. Or, in simpler words, instead of an address field, the immediate-mode instruction consists of an operand field. An operand field contains the actual operand that is to be used in conjunction with an operation that is determined in the given instruction. The immediate-mode instructions help initialize registers to a certain constant value.

Register Mode

In the register mode, the operands exist in those registers that reside within a CPU. In this case, we select a specific register from a certain register field in the given instruction. The k-bit field is capable of determining one 2k register.

Register Indirect Mode

In the register indirect mode, the instruction available to us defines that particular register in the CPU whose contents provides the operand's address in the memory. In simpler words, any selected register would include the address of an operand instead of the operand itself.

The reference to a register is equivalent to specifying any memory address. The pros of using this type of instruction are that an instruction's address field would make use of fewer bits to select a register than would be require when someone wants to directly specify a memory address.

Autodecrement or the Autoincrement Mode

The Autodecrement or Autoincrement mode is very similar to the register indirect mode. The only exception is that the register is decremented or incremented before or after its value is used to access memory. When the address stored in the register defines a data table in memory, it is very crucial to decrement or increment the register after accessing the table every time. It can be obtained using the decrement or increment instruction.

Direct Address Mode

In the direct address mode, the address part of the instruction is equal to the effective address. The operand would reside in memory, and the address here is given directly by the instruction's address field. The address field would specify the actual branch address in a branch-type instruction.

Indirect Address Mode

In an indirect address mode, the address field of an available instruction gives that address in which the effective address gets stored in memory. The control fetches the instruction available in the memory and then uses its address part in order to (again) access memory to read its effective address.

Indexed Addressing Mode

In the indexed addressing mode, the content of a given index register gets added to an instruction's address part so as to obtain the effective address. Here, the index register refers to a special CPU register that consists of an index value. An instruction's address field defines the beginning address of any data array present in memory.

Unit-II

Arithmetic and logic unit

2.1. Carry Look Ahead Adders

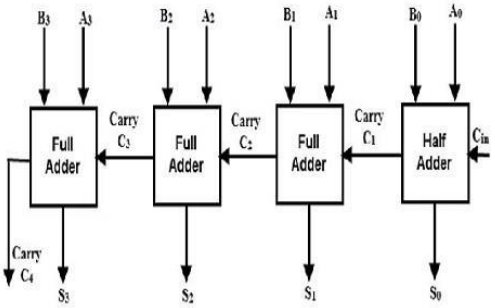
A digital computer must contain circuits which can perform arithmetic operations such as addition, subtraction, multiplication, and division. Among these, addition and subtraction are the basic operations whereas multiplication and division are the repeated addition and subtraction respectively. To perform these operations 'Adder circuits' are implemented using basic logic gates. Adder circuits are evolved as Half-adder, Full-adder, Ripple-carry Adder, and Carry Look-ahead Adder.

Among these Carry Look-ahead Adder is the faster adder circuit. It reduces the propagation delay, which occurs during addition, by using more complex hardware circuitry. It is designed by transforming the ripple-carry Adder circuit such that the carry logic of the adder is changed into two-level logic.

In parallel adders, carry output of each full adder is given as a carry input to the next higher-order state. Hence, these adders it is not possible to produce carry and sum outputs of any state unless a carry input is available for that state.

So, for computation to occur, the circuit has to wait until the carry bit propagated to all states This induces carry propagation delay in the circuit.

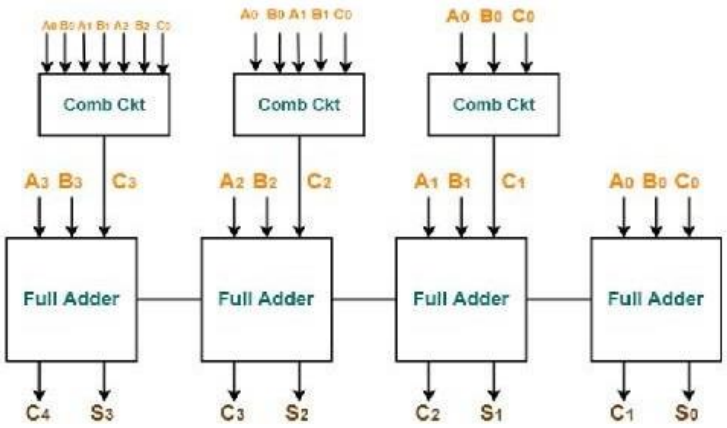
Consider the 4-bit ripple carry adder circuit above. Here the sum S3 can be produced as soon as the inputs A3 and B3 are given. But carry C3 cannot be computed until the carry bit C2 is applied whereas C2 depends on C1. Therefore to produce final steady-state results, carry must propagate through all the states. This increases the carry propagation delay of the circuit.



The propagation delay of the adder is calculated as "the propagation delay of each gate times the number of stages in the circuit". For the computation of a large number of bits, more stages have to

be added, which makes the delay much worse. Hence, to solve this situation, Carry Look-ahead Adder was introduced.

To understand the functioning of a Carry Look-ahead Adder, a 4-bit Carry Look-ahead Adder is described below.



In this adder, the carry input at any stage of the adder is independent of the carry bits generated at the independent stages. Here the output of any stage is dependent only on the bits which are added in the previous stages and the carry input provided at the beginning stage. Hence, the circuit at any stage does not have to wait for the generation of carry-bit from the previous stage and carry bit can be evaluated at any instant of time.

Truth Table of Carry Look-ahead Adder

For deriving the truth table of this adder, two new terms are introduced – Carry generate and carry propagate. Carry generate Gi =1 whenever there is a carry Ci+1 generated. It depends on Ai and Bi inputs. Gi is 1 when both Ai and Bi are 1. Hence, Gi is calculated as Gi = Ai.Bi.

Carry propagated Pi is associated with the propagation of carry from Ci to Ci+1. It is calculated as Pi =Ai. Using the Gi and Pi terms the Sum Si and Carry Ci+1 are given as below –

Si = Pi $\oplus$ Gi.

Ci+1 = Ci.Pi +Gi.

Therefore, the carry bits C1, C2, C3, and C4 can be calculated as

C1 = C0.P0+G0.

C2 = C1.P1+G1 = (C0.P0+G0).P1+G1.

C3 = C2.P2+G2 = (C1.P1+G1).P2+G2.

C4 = C3.P3+G3 = C0.P0.P1.P2.P3 + P3.P2.P1.G0 + P3.P2.G1 + G2.P3 + G3.

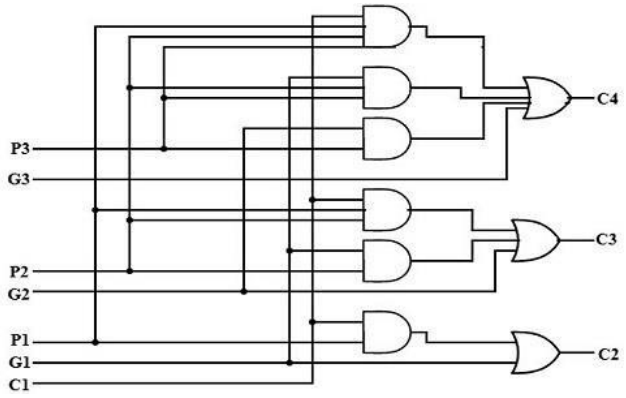
It can be observed from the equations that carry Ci+1 only depends on the carry C0, not on the intermediate carry bits.

Ai ⊕ Bi. The truth table of this adder can be derived from modifying the truth table of a full adder.

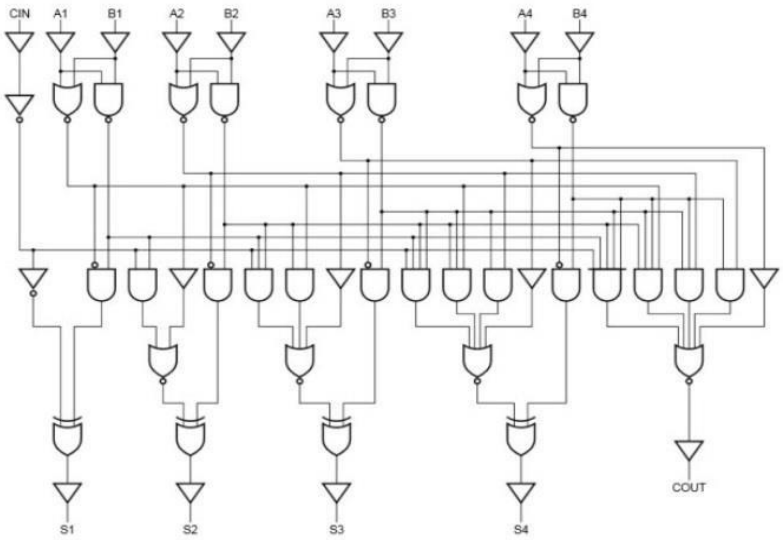
A	B	Ci	C i +1	Condition
0	0	0	0	No carry generate
0	0	1	0	
0	1	0	0	
0	1	1	1	No carry propagate
1	0	0	0	
1	0	1	1	
1	1	0	1	Carry generate
1	1	1	1	

Circuit Diagram

The above equations are implemented using two-level combinational circuits along with AND, OR gates, where gates are assumed to have multiple inputs.



The Carry Look-ahead Adder circuit for 4-bit is given below



Advantages of Carry Look-ahead Adder

In this adder, the propagation delay is reduced. The carry output at any stage is dependent only on the initial carry bit of the beginning stage. Using this adder it is possible to calculate the intermediate results. This adder is the fastest adder used for computation.

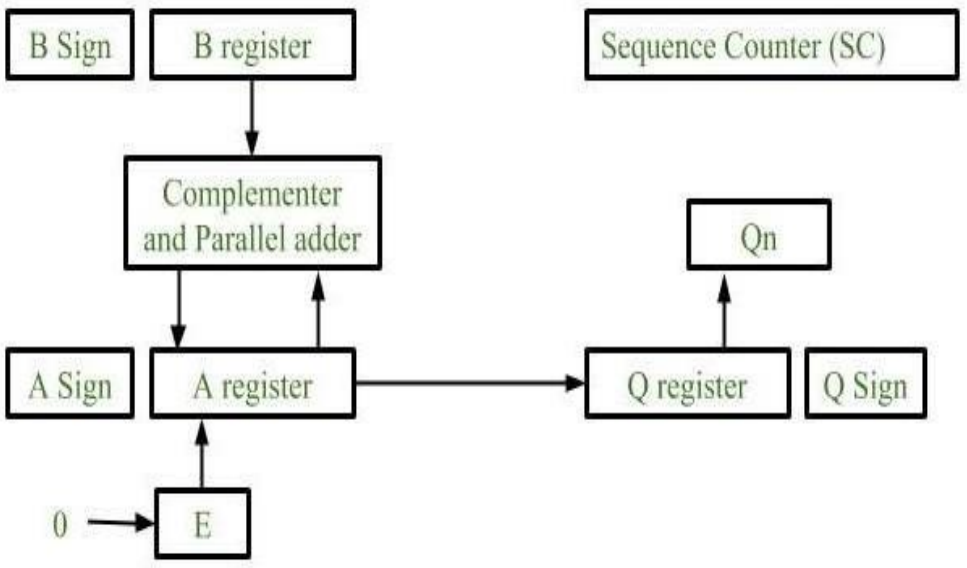
2.2. Multiplication Algorithm:

- Multiplication of two fixed-point binary numbers in signed-magnitude representation is done with process of successive shift and adds operations. This process is best illustrated with a numerical example as follows:

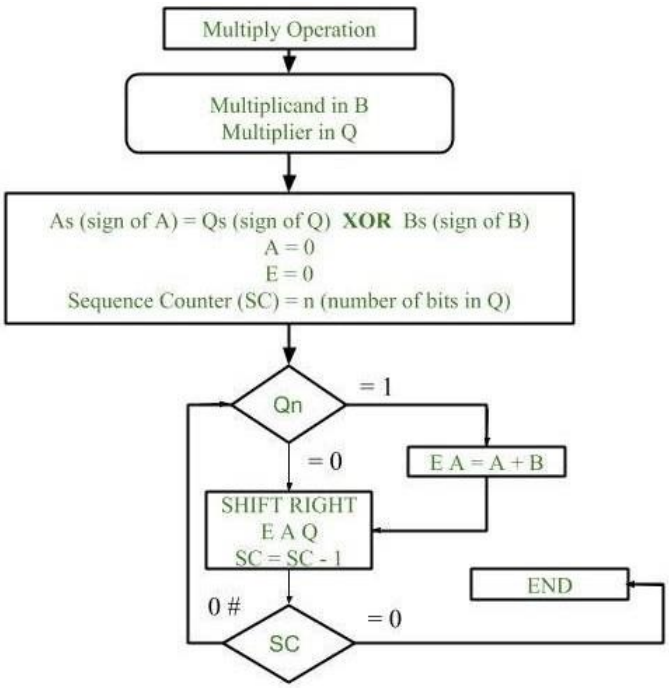
```
   10111 (Multiplicand)
x  10011 (Multiplier)
  ------
   10111
   10111
  00000
 00000
10111
---------
011011010    (Product)
---------
```

2.2.1. Hardware Implementation for Signed-Magnitude Data Multiplication

Following components are required for the Hardware Implementation of multiplication algorithm :



Flowchart of Multiplication:



- Initially, the multiplicand is in register B and the multiplier in Q.
- Initially multiplicand is stored in B register and multiplier is stored in Q register.
- Sign of registers B (Bs) and Q (Qs) are compared using XOR functionality (i.e., if both the signs are alike, output of XOR operation is 0 unless 1) and output stored in As (sign of A register). Note: Initially 0 is assigned to register A and E flip flop. Sequence counter is initialized with value n, n is the number of bits in the Multiplier.

- Now least significant bit of multiplier is checked. If it is 1 add the content of register A with Multiplicand (register B) and result is assigned in A register with carry bit in flip flop E. Content of E A Q is shifted to right by one position, i.e., content of E is shifted to most significant bit (MSB) of A and least significant bit of A is shifted to most significant bit of Q.
- If Qn = 0, only shift right operation on content of E A Q is performed in a similar fashion. Content of Sequence counter is decremented by 1.
- Check the content of Sequence counter (SC), if it is 0, end the process and the final product is present in register A and Q, else repeat the process.

2.2.2. Booth’s Algorithm for Signed Magnitude Data

Booth's algorithm is a fast multiplication algorithm that is used to multiply two signed magnitude binary numbers. It is a parallel algorithm that uses two's complement representations and shift-and-add operations to perform multiplication. The algorithm is named after Andrew Donald Booth, who developed it in 1951.

The basic idea behind Booth's algorithm is to take advantage of the representation of signed magnitude binary numbers to perform multiplication efficiently. The algorithm starts by multiplying the two numbers one bit at a time, shifting the intermediate result and adding or subtracting the multiplicand as necessary. The result of the multiplication is then accumulated in a register.

Here is an example of how Booth's algorithm works for the multiplication of two signed magnitude:

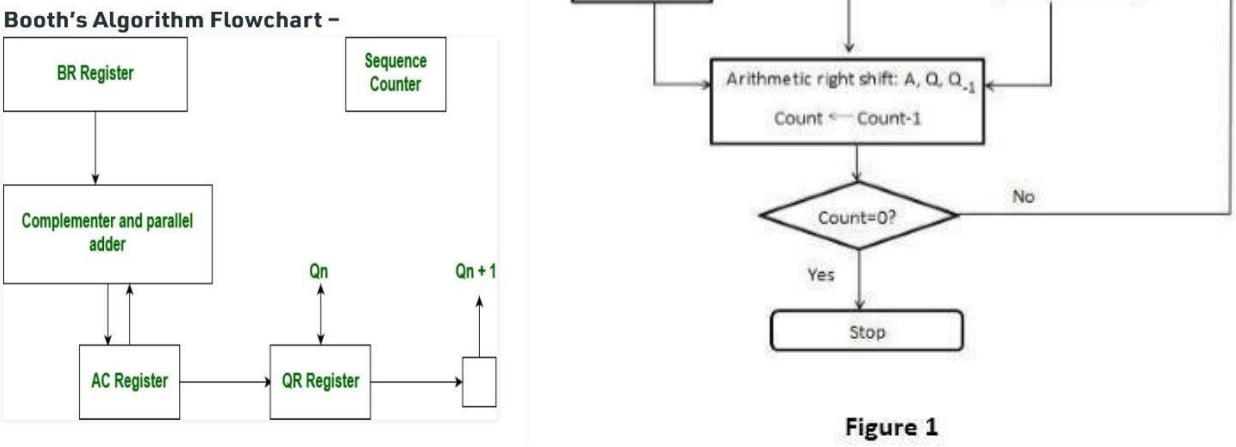
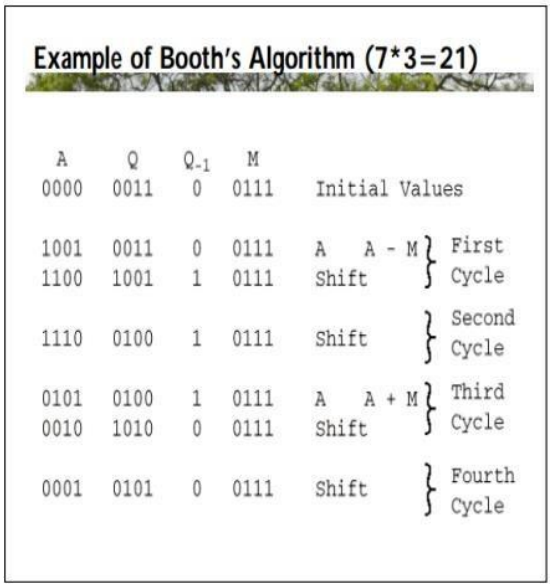


Figure 1

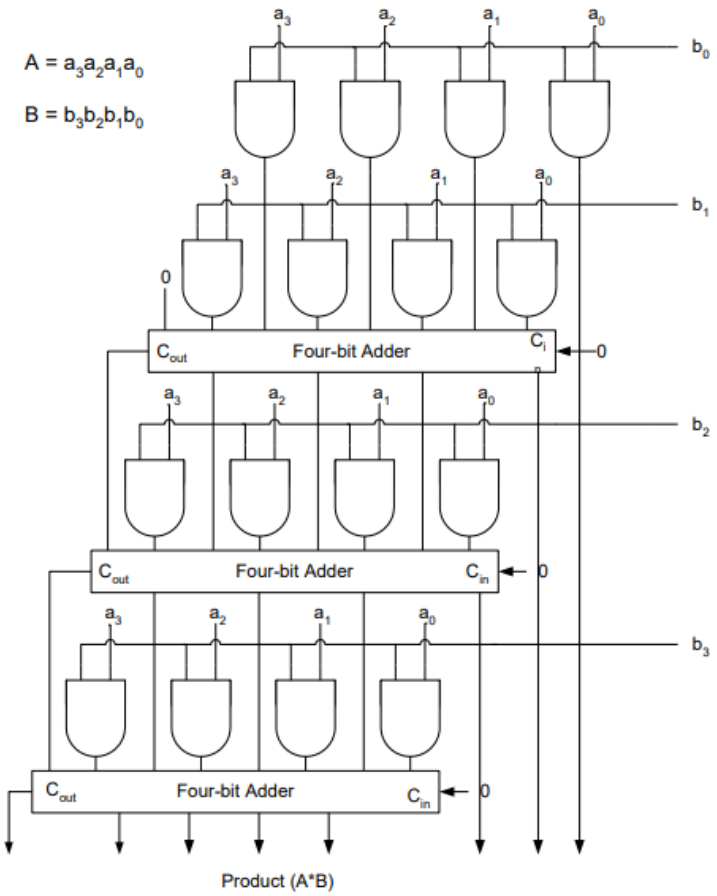


Array Multiplier:

An array multiplier is a digital circuit that is used to perform the multiplication of two arrays of numbers. Array multipliers are used in various applications, including digital signal processing, image processing, and cryptography. Array multipliers can be implemented in several ways, including using conventional digital circuits such as full adders and partial product generators, or using specialized hardware such as field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs).

Shift and Add Multiplier

It is similar to the normal multiplication process, which we do in mathematics, from array multiplier flow chat where X = Multiplicand; Y = Multiplier; A = Accumulator, Q = Quotient. Firstly Q is checked if it's 1 or no if it is 1 then add A and B and shift A_Q arithmetic right, else if it is not 1 directly shift A_Q arithmetic right and decrement N by 1, in the next step check if N is 0 or no. If N not 0 repeats from Q=0 step else terminate the process.



Construction and Working of a 4×4 Array Multiplier

The design structure of the array Multiplier is regular, it is based on the add shift algorithm principle.

Partial product = the multiplicand * multiplier bit ..... (2)

where AND gates are used for the product, the summation is done using Full Adders and Half Adders where the partial product is shifted according to their bit orders. In an n*n array multiplier, n*n AND gates compute the partial products and the addition of partial products can be performed by using n* (n – 2) Full adders and n Half adders. The 4×4 array multiplier shown has 8 inputs and 8 outputs

Advantages of 4×4 Array Multiplier:

- Minimum complexity
- Easily scalable
- Easily pipelined
- Regular shape, easy to place and route

Disadvantages of 4×4 Array Multiplier

- High power consumption
- More digital gates resulting in large areas.

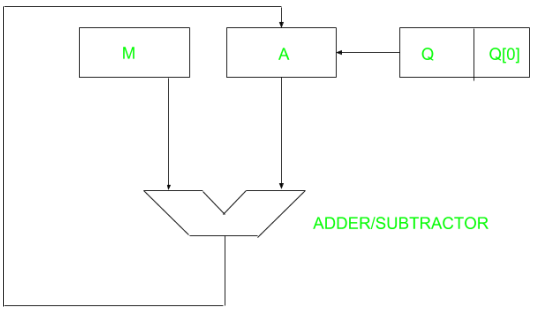
Applications of 4×4 Array Multiplier

- Array multiplier is used to perform the arithmetic operation, like filtering, Fourier transform, image coding.
- High-speed operation.

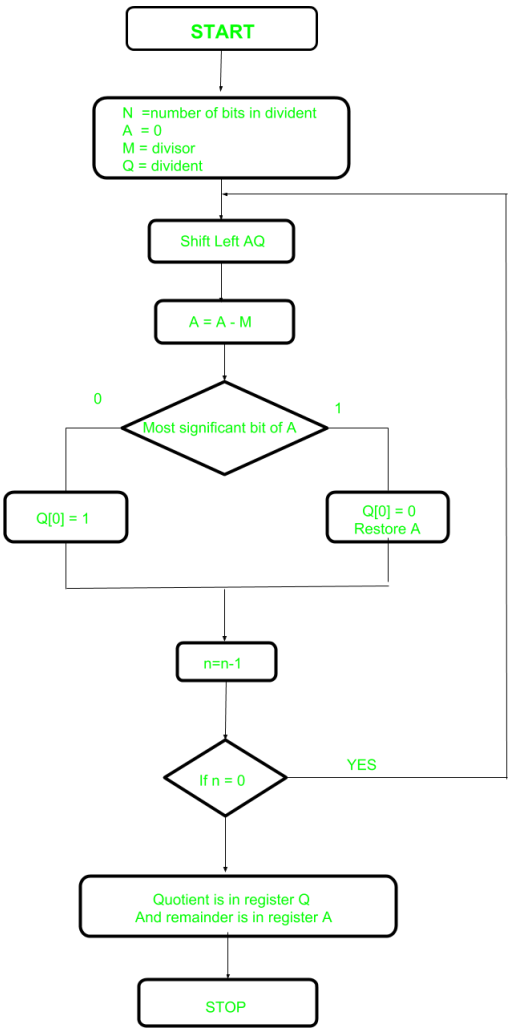
2.3. Division Algorithm

A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type slow algorithm and fast algorithm. Slow division algorithm are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt.

In this article, will be performing restoring algorithm for unsigned integer. Restoring term is due to fact that value of register A is restored after each iteration.



Here, register Q contain quotient and register A contain remainder. Here, n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.



Step-1: First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)

Step-2: Then the content of register A and Q is shifted left as if they are a single unit

Step-3: Then content of register M is subtracted from A and result is stored in A

Step-4: Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M

Step-5: The value of counter n is decremented

Step-6: If the value of n becomes zero we get of the loop otherwise we repeat from step 2

Step-7: Finally, the register Q contain the quotient and A contain remainder

2.4. Floating Point Arithmetic Operations:

Arithmetic operations on floating point numbers consist of addition, subtraction, multiplication and division. The operations are done with algorithms similar to those used on sign magnitude integers (because of the similarity of representation).

example, only add numbers of the same sign. If the numbers are of opposite sign, must do subtraction.

ADDITION

Example on decimal value given in scientific notation:

3.25 x 10 ** 3

+ 2.63 x 10 ** -1

---

first step: align decimal points

second step: add

3.25 x 10 ** 3

+ 0.000263 x 10 ** 3

3.250263 x 10 ** 3 (presumes use of infinite precision, without regard for accuracy)

third step: normalize the result (already normalized!)

Example on floating pt. value given in binary:

.25 = 0 01111101 00000000000000000000000

100 = 0 10000101 10010000000000000000000

To add these fl. pt. representations,

step 1: align radix points

shifting the mantissa left by 1 bit decreases the exponent by 1

shifting the mantissa right by 1 bit increases the exponent by 1

we want to shift the mantissa right, because the bits that fall off the end should come from the least significant end of the mantissa

-> choose to shift the .25, since we want to increase it's exponent.

-> shift by 10000101

-01111101

00001000 (8) places.

0 01111101 00000000000000000000000 (original value)

0 01111110 10000000000000000000000 (shifted 1 place)

(note that hidden bit is shifted into msb of mantissa)

0 01111111 01000000000000000000000 (shifted 2 places)

0 10000000 00100000000000000000000 (shifted 3 places)

0 10000001 00010000000000000000000 (shifted 4 places)

0 10000010 00001000000000000000000 (shifted 5 places)

0 10000011 00000100000000000000000 (shifted 6 places)

0 10000100 00000010000000000000000 (shifted 7 places)

0 10000101 00000001000000000000000 (shifted 8 places)

step 2: add (don't forget the hidden bit for the 100)

0 10000101 1.10010000000000000000000 (100)

\+ 0 10000101 0.00000001000000000000000 (.25)

---

0 10000101 1.10010001000000000000000

step 3: normalize the result (get the "hidden bit" to be a 1) It already is for this example.

result is 0 10000101 10010001000000000000000

SUBTRACTION

Same as addition as far as alignment of radix points Then the algorithm for subtraction of sign mag. numbers takes over.

before subtracting,

compare magnitudes (don't forget the hidden bit!)

change sign bit if order of operands is changed.

don't forget to normalize number afterward.

MULTIPLICATION

Example on decimal values given in scientific notation:

3.0 x 10 ** 1

+ 0.5 x 10 ** 2

Algorithm: multiply mantissas

add exponents

3.0 x 10 ** 1

+ 0.5 x 10 ** 2

1.50 x 10 ** 3

Example in binary: Consider a mantissa that is only 4 bits.

0 10000100 0100

x 1 00111100 1100

---

mantissa multiplication:
(don't forget hidden bit)

```
 1.0100
x 1.1100
------
 00000
 00000
 10100
 10100
 10100
--------
1000110000
```

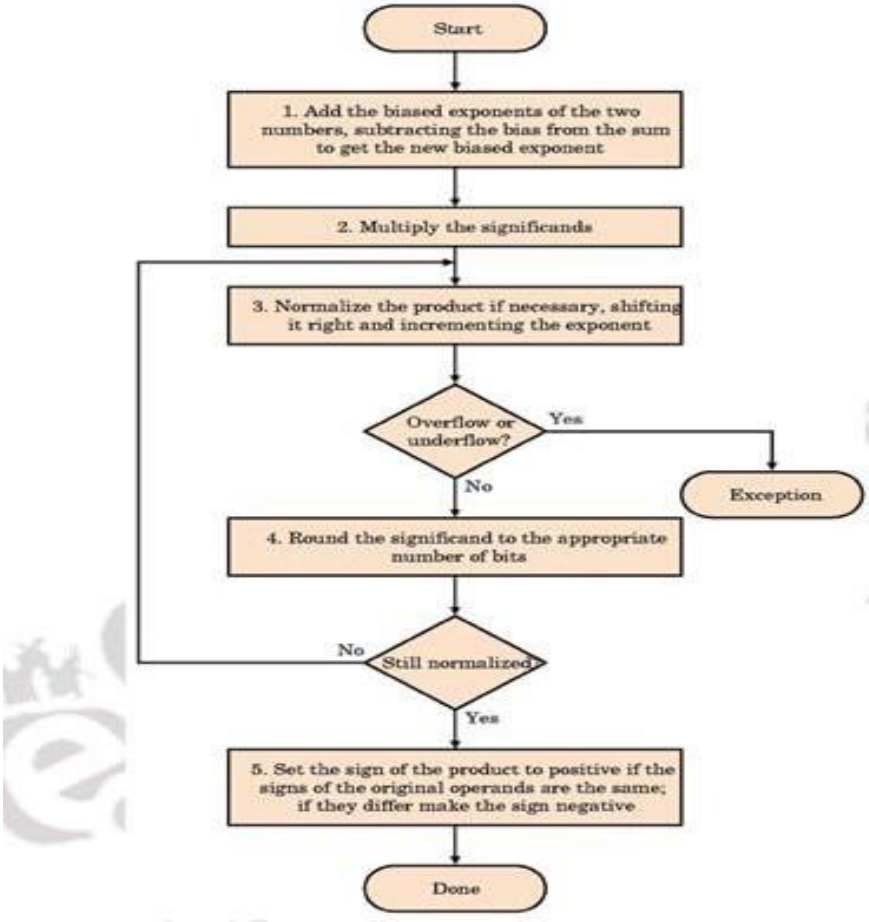
DIVISION

It is similar to multiplication.

do unsigned division on the mantissas (don’t forget the hidden bit)

subtract TRUE exponents

Flowchart :



2.5. IEEE standard for Floating Point Numbers :

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**. The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.

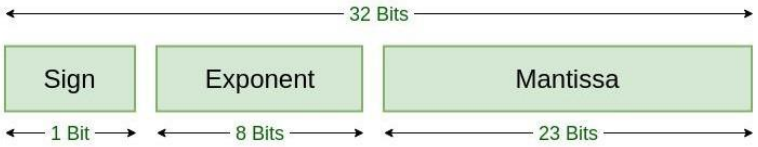
here are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE 754 has 3 basic components:

The Sign of Mantissa – This is as simple as the name. 0 represents a positive number while 1 represents a negative number.

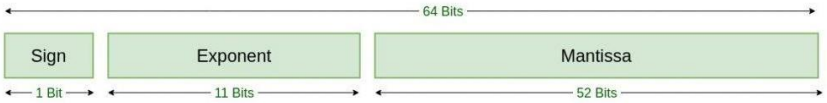
The Biased exponent – The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.

The Normalised Mantissa – The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

IEEE 754 numbers are divided into two based on the above three components: single precision and double precision.



Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

Example –

85.125

85 = 1010101

0.125 = 001

85.125 = 1010101.001

=1.010101001 x 2^6

sign = 0

1. Single precision:

biased exponent 127+6=133

133 = 10000101

Normalised mantisa = 010101001

we will add 0's to complete the 23 bits

The IEEE 754 Single precision is:

= 0 10000101 01010100100000000000000

This can be written in hexadecimal form **42AA4000**

2. Double precision:

biased exponent 1023+6=1029

1029 = 10000000101

Normalised mantisa = 010101001

we will add 0's to complete the 52 bits

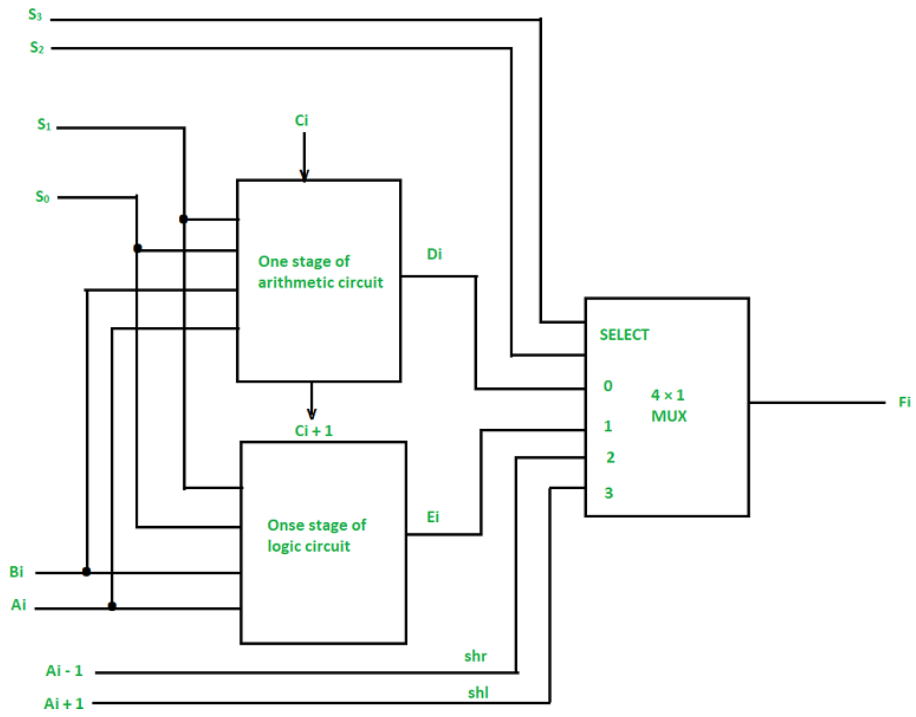
The IEEE 754 Double precision is:

= 0 10000000101 0101010010000000000000000000000000000000000000000000

This can be written in hexadecimal form **4055480000000000**

2.6. ALU Design:

Arithmetic Logic Shift Unit (ALSU) It is a digital circuit that performs logical, arithmetic, and shift operations. Rather than having individual registers calculating the micro operations directly, the computer deploys a number of storage registers which is connected to a common operational unit known as an arithmetic logic unit or ALU.



We can combine and make one ALU with common selection variables by adding arithmetic, logic, and shift circuits. We can see the, One stage of an arithmetic logic shift unit in the diagram below. Some particular micro operations are selected through the inputs S1 and S0.

4 x 1 multiplexer at the output chooses between associate arithmetic output between Ei and a logic output in Hi. The data in the multiplexer are selected through inputs S3 and S2 and the other two data inputs to the multiplexer obtain the inputs Ai – 1 for the ***shr*** operation and Ai + 1 for the ***shl*** operation.

Note: The output carry Ci + 1 of a specified arithmetic stage must be attached to the input carry Ci of the next stage in the sequence.

The circuit whose one stage is given in the below diagram provides 8 arithmetic operations, 4 logic operations, and 2 shift operations, and Each operation is selected by the 5 variables S3, S2, S1, S0, and Cin.

The below table shows the 14 operations perform by the Arithmetic Logic Unit:

The first 8 are arithmetic operations which are selected by S3 S2 = 00

The next 4 are logic operations which are selected by S3 S2 = 01

The last two are shift operations which are selected by S3 S2 = 10 & 11

Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + B'	Subtract with borrow
0	0	1	0	1	F = A + B' + 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F= A	Transfer A
0	1	0	0	×	F = A ^ B	AND
0	1	0	1	×	F = A v B	OR
0	1	1	0	×	F = A XOR B	XOR
0	1	1	1	×	F = A'	Complement A
1	0	×	×	×	F = shr A	Shift right A into F
1	1	×	×	×	F = shl A	Shift left A into F

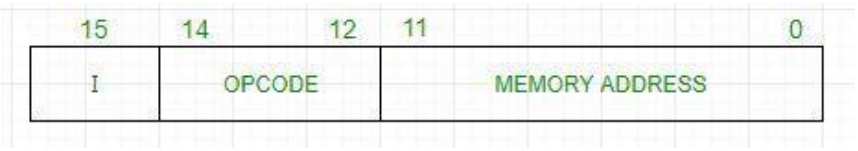
Unit-III

Control Unit

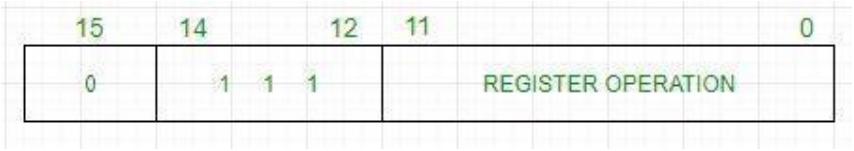
3.1. Instruction Types

The basic computer has 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

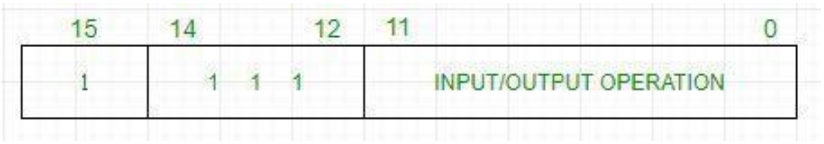
Memory Reference – These instructions refer to memory address as an operand. The other operand is always accumulator. Specifies 12-bit address, 3-bit opcode (other than 111) and 1-bit addressing mode for direct and indirect addressing.



Register Reference – These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.



Input/Output – These instructions are for communication between computer and outside environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.



3.2. Instruction Formats

A computer performs a task based on the instruction provided. Instruction in computers comprises groups called fields. These fields contain different information as for computers everything is in 0 and 1 so each field has different significance based on which a CPU decides what to perform. The most common fields are:

- **Operation field** specifies the operation to be performed like addition.
- **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

.

Instruction is of variable length depending upon the number of addresses it contains. Generally, CPU organization is of three types based on the number of address fields:

- Single Accumulator organization
- General register organization
- Stack organization

In the first organization, the operation is done involving a special register called the accumulator. In second on multiple registers are used for the computation purpose. In the third organization the work on stack basis operation due to which it does not contain any address field. Only a single organization doesn't need to be applied, a blend of various organizations is mostly what we see generally. Based on the number of address, instructions are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

i. Zero Address Instructions

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

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

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

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

ii. One Address Instructions –

This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

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

AC is accumulator

M[] is any memory location

M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

iii. Two Address Instructions –

This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent address.

Here destination address can also contain operand.

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

iv. Three Address Instructions –

This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

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

R1, R2 are registers

M[] is any memory location

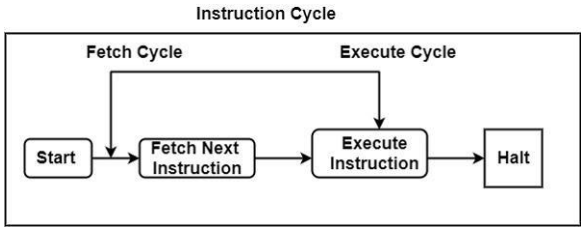
ADD	R1, A, B	R1 = M[A] + M[B]
ADD	R2, C, D	R2 = M[C] + M[D]
MUL	X, R1, R2	M[X] = R1 * R2

3.3. Instruction Cycle and Sub Cycles

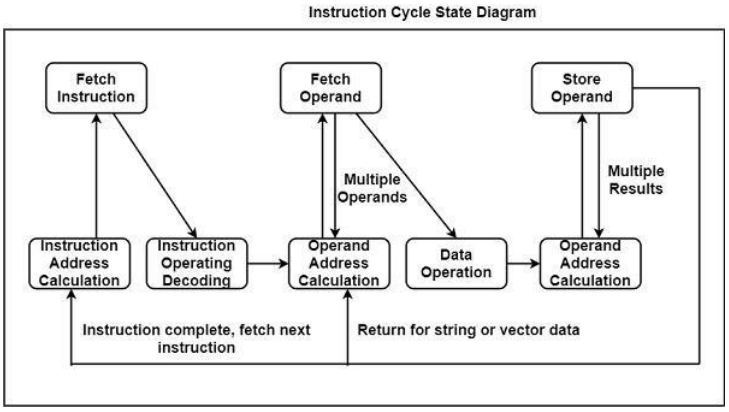
A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction.

Instruction Cycle is subdivided into following subcycles:

1. **Fetch instruction from memory.**(At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC))
2. **Decode the instruction**.(Decoder circuit examines the opcode of the instruction.Result is selecting a unique decoder output line.)
3. **Read the effective address from memory**.
4. **Execute the instruction**(Microcode for the instruction, selected by the decoder output line, is executed by the ALU.)



State Diagram for Instruction Cycle



Instruction Address Calculation − The address of the next instruction is computed. A permanent number is inserted to the address of the earlier instruction.

Instruction Fetch − The instruction is read from its specific memory location to the processor.

Instruction Operation Decoding − The instruction is interpreted and the type of operation to be implemented and the operand(s) to be used are decided.

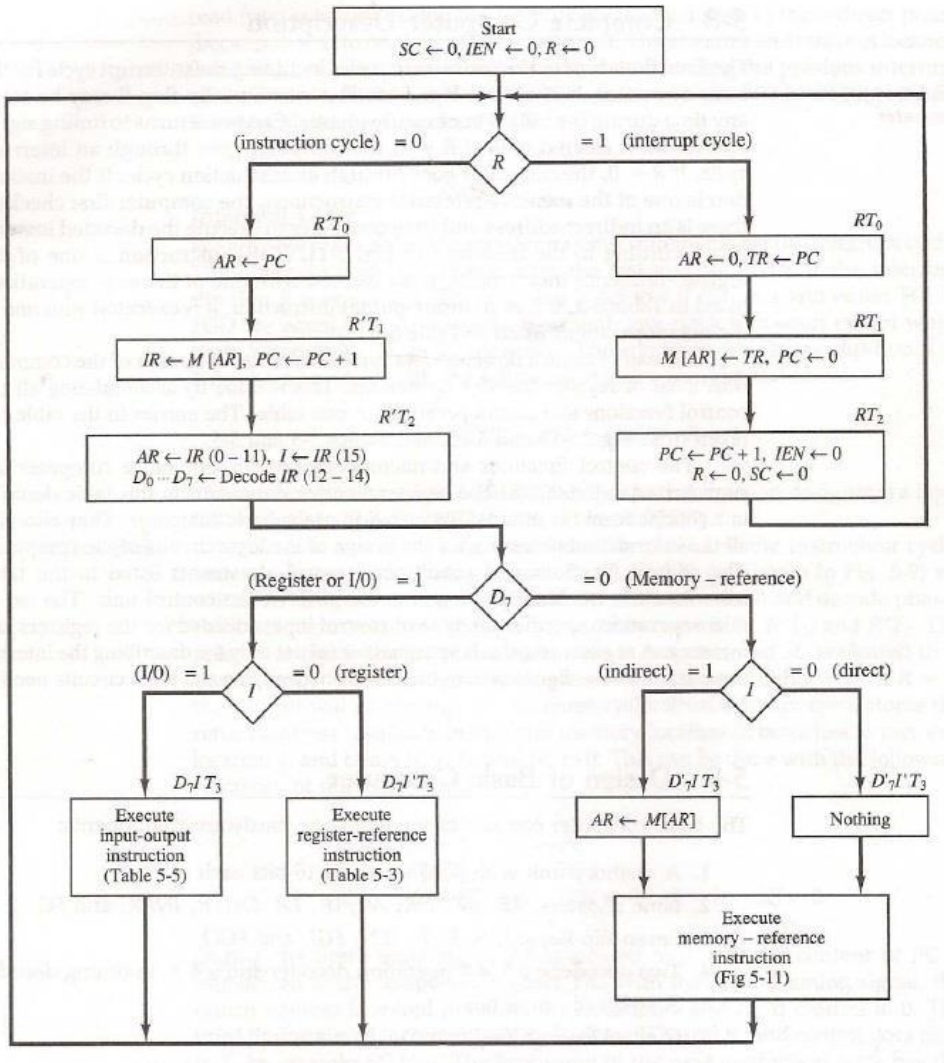
Operand Address Calculation − The address of the operand is evaluated if it has a reference to an operand in memory or is applicable through the Input/Output.

Operand Fetch − The operand is read from the memory or the I/O.

Data Operation − The actual operation that the instruction contains is executed.

Store Operands − It can store the result acquired in the memory or transfer it to the I/O.

Complete Computer Operation Flowchart:

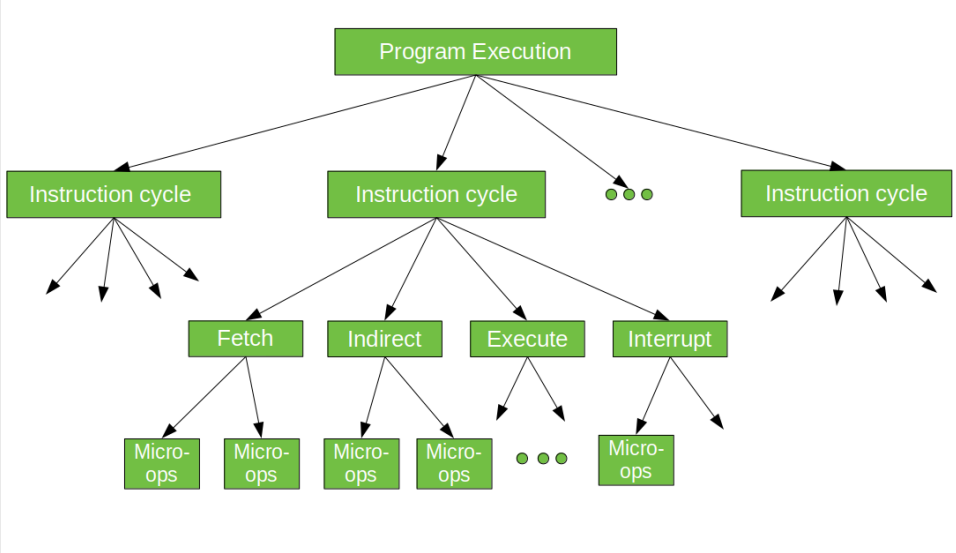


You may be speculative however the central processor is programmed. It contains a special register — the instruction register — whose bit pattern determines what the central processor unit can do. Once that action has been completed, the bit pattern within the instruction register may be modified, and also the central processor unit can perform the operation nominative by this next bit pattern.
Since directions are simply bit patterns, they will be kept in memory. The instruction pointer register continuously has the memory address of (points to) the next instruction to be executed. so as for the management unit to execute this instruction, it's derived into the instruction register. the case is as follows:
A sequence of instructions is stored in memory.

1. The memory address wherever the first instruction is found is copied to the instruction pointer.
2. The CPU sends the address within the instruction pointer to memory on the address bus.
3. The CPU sends a "read" signal to the control bus.
4. Memory responds by sending a copy of the state of the bits at that memory location on the data bus, that the CPU then copies into its instruction register.
5. The instruction pointer is automatically incremented to contain the address of the next instruction in memory.
6. The CPU executes the instruction within the instruction register.
7. Go to step 3

3.4. Micro-operations

Operation of a computer consists of a sequence of instruction cycles, with one machine instruction per cycle. Each instruction cycle is made up of a number of smaller units – Fetch, Indirect, Execute and Interrupt cycles. Each of these cycles involves series of steps, each of which involves the processor registers. These steps are referred as micro-operations. the prefix micro refers to the fact that each of the step is very simple and accomplishes very little. Figure below depicts the concept being discussed here.



3.5. Execution of a Complete Instructions:

We have discussed about four different types of basic operations:

- Fetch information from memory to CPU
- Store information to CPU register to memory
- Transfer of data between CPU registers.
- Perform arithmetic or logic operation and store the result in CPU registers.

To execute a complete instruction, we need to take help of these basic operations and we need to execute this operation in some particular order.

As for example, consider the instruction: “Add contents of memory location NUM to the contents of register R1 and store the result in register R1.” For simplicity, assume that the address NUM is given explicitly in the address field of the instruction. That is, in this instruction, direct addressing mode is used.

Execution of this instruction requires the following action :

1. Fetch instruction
2. Fetch first operand (Contents of memory location pointed atby the address field of the instruction)
3. Perform addition
4. Load the result into R1.

Following sequence of control steps are required to Implement

Steps	Actions
1.	PC_{out}, MAR_{in}, Read, Clear Y, Set carry -in to ALU, Add, Z_{in}
2.	Z_{out}, PC_{in}, Wait For MFC
3.	MDR_{out}, Ir_{in}
4.	Address-field- of-IR_{out}, MAR_{in}, Read
5.	$R1_{out}$, Y_{in}, Wait for MFC
6.	MDR_{out}, Add, Z_{in}
7.	Z_{out}, $R1_{in}$
8.	END

THANKS TO SECTION-11 & SECTION 24