COMPUTER ARCHITECTURE & ORGANIZATION

Lecture Notes

III B.Tech – I Semester – R20 Regulation

ELECTRONICS & COMMUNICATION ENGINEERING

Prepared by;

P. IMRAN KHAN M.Tech.(Ph.D).,

Assistant Professor

Department of Electronics and Communication Engineering

St.JOHNS COLLEGE OF ENGINEERING & TECHNOLOGY

(Affiliated to Jawaharlal Nehru Technological University, Anantapur, A.P.)

Yerrakota, Yemmiganur-518360, Kurnool(Dist), A.P.

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, ANANTAPUR

B.Tech (ECE) – III-I Sem

L	T	P	C
3	0	0	3

COMPUTER ARCHITECTURE & ORGANIZATION

- (20A04504a)

Course Objectives:

The purpose of the course is to introduce principles of computer organization and the basic architectural concepts.

Course Outcomes:

- *Understand the basics of instructions sets and their impact on processor design.*
- *Demonstrate an understanding of the design of the functional units of a digital computer system.*
- *Evaluate cost performance and design trade-offs in designing and constructing a computer processor including memory.*
- *Design a pipeline for consistent execution of instructions with minimum hazards.*
- *Recognize and manipulate representations of numbers stored in digital computers.*

UNIT I

Digital Computers: Introduction, Block diagram of Digital Computer, Definition of Computer Organization, Computer Design and Computer Architecture.

Register Transfer Language and Micro operations: Register Transfer language, Register Transfer, Bus and memory transfers, Arithmetic Micro operations, logic micro operations, shift micro operations, Arithmetic logic shift unit.

Basic Computer Organization and Design: Instruction codes, Computer Registers Computer instructions, Timing and Control, Instruction cycle, Memory Reference Instructions, Input – Output and Interrupt.

UNIT II

Micro programmed Control: Control memory, Address sequencing, micro program example, design of control unit.

Central Processing Unit: General Register Organization, Instruction Formats, Addressing modes, Data Transfer and Manipulation, Program Control.

UNIT III

Data Representation: Data types, Complements, Fixed Point Representation, Floating Point Representation.

Computer Arithmetic: Addition and subtraction, multiplication Algorithms, Division Algorithms, Floating – point Arithmetic operations. Decimal Arithmetic unit, Decimal Arithmetic operations.

UNIT IV

Input-Output Organization: Input-Output Interface, Asynchronous data transfer, Modes of Transfer, Priority Interrupt Direct memory Access.

Memory Organization: Memory Hierarchy, Main Memory, Auxiliary memory, Associate Memory, Cache Memory.

UNIT V

Reduced Instruction Set Computer: CISC Characteristics, RISC Characteristics. Pipeline and Vector Processing: Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction Pipeline, RISC Pipeline, Vector Processing, Array Processor. Multi Processors: Characteristics of Multiprocessors, Interconnection Structures, Interprocessor arbitration, Interprocessor communication and synchronization, Cache Coherence.

Textbook:

1. Computer System Architecture – M. Moris Mano, Third Edition, Pearson/PHI.

References:

1. Computer Organization – Car Hamacher, ZvonksVranesic, SafeaZaky, V th Edition, McGraw Hill.
2. Computer Organization and Architecture – William Stallings Sixth Edition, Pearson/PHI.
3. Structured Computer Organization – Andrew S. Tanenbaum, 4th Edition, PHI/Pearson.

UNIT – 01 – Part - A

DIGITAL COMPUTERS

Digital Computers:

A Digital computer can be considered as a digital system that performs various computational tasks.

The **digital computer** is a digital system that performs various computational tasks. The word **digital** implies that the information in the computer is represented by variables that take a limited number of discrete values. These values are processed internally by components that can maintain a limited number of discrete states.

The decimal digits 0, 1, 2, ..., 9, for example, provide 10 discrete values. The first electronic digital computer, developed in the late 1940s, was used primarily for numerical computations and the discrete elements were the digits. From this application the term **digital** computer emerged.

In practice, digital computers function more reliably if only two states are used. Because of the physical restriction of components, and because human logic tends to be binary (i.e. true or false, yes or no statements), digital components that are constrained to take discrete values are further constrained to take only two values and are said to be **binary**.

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.

The first electronic digital computer was developed in the late 1940s and was used primarily for numerical computations.

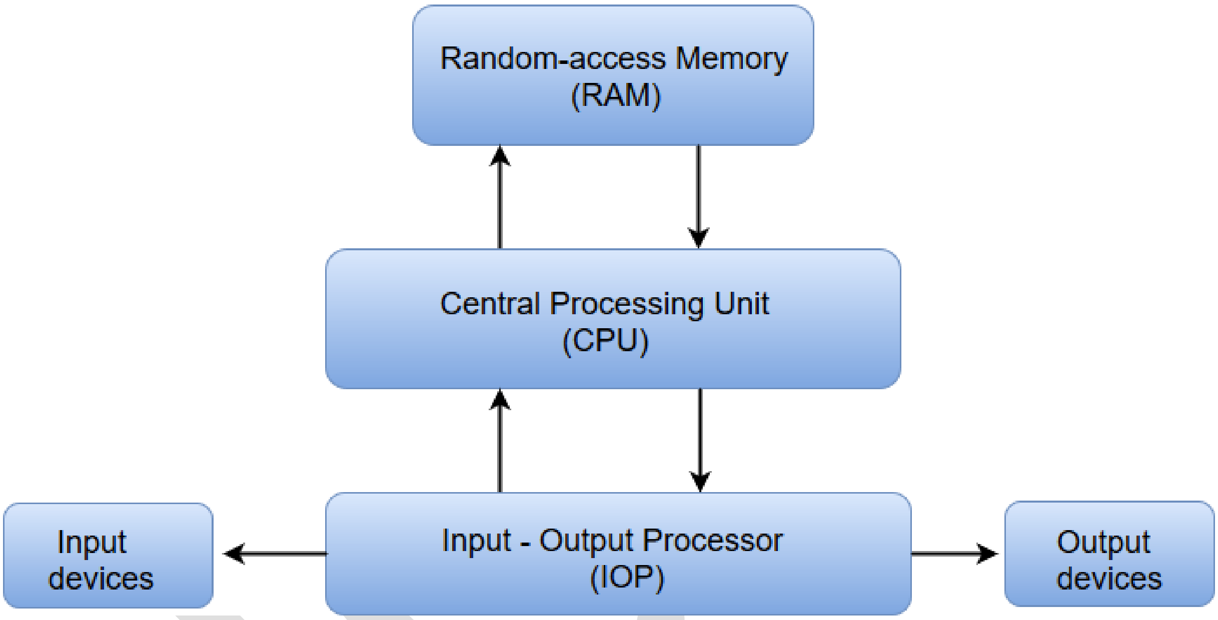
By convention, the digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit.

A computer system is subdivided into two functional entities: Hardware and Software.

The hardware consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.

The software of the computer consists of the instructions and data that the computer manipulates to perform various data-processing tasks.

Block diagram of a digital computer:

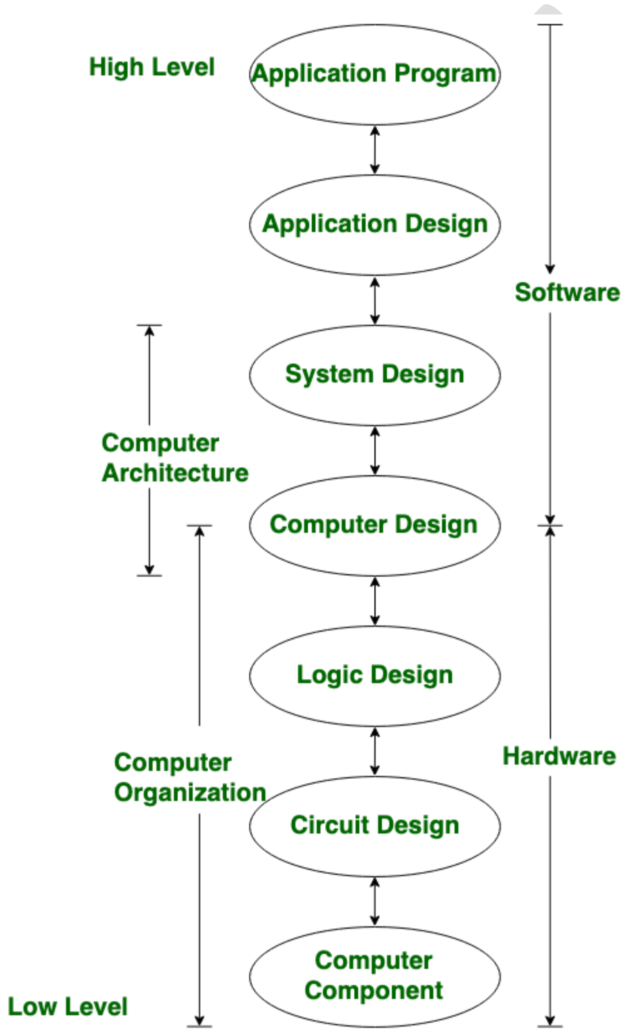


- The Central Processing Unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and a control circuit for fetching and executing instructions.
- The memory unit of a digital computer contains storage for instructions and data.
- The Random Access Memory (RAM) for real-time processing of the data.
- The Input-Output devices for generating inputs from the user and displaying the final results to the user.
- The Input-Output devices connected to the computer include the keyboard, mouse, terminals, magnetic disk drives, and other communication devices.

Computer Architecture:

Computer Architecture is a functional description of requirements and design implementation for the various parts of a computer. It deals with the functional behavior of computer systems. It comes before the computer organization while designing a computer.

Architecture describes what the computer does.



Computer System

Computer Organization:

Computer Organization **comes** after the decision of Computer Architecture first. Computer Organization is how operational attributes are linked together and contribute to realizing the architectural specification. Computer Organization deals with a structural relationship.

The organization describes how it does it.

Difference between Computer Architecture and Computer Organization:

Computer Architecture	Computer Organization
Architecture describes what the computer does.	The Organization describes how it does it.
Computer Architecture deals with the functional behavior of computer systems.	Computer Organization deals with a structural relationship.
In the above figure, it's clear that it deals with high-level design issues.	In the above figure, it's also clear that it deals with low-level design issues.
Architecture indicates its hardware.	Where Organization indicates its performance.
As a programmer, you can view architecture as a series of instructions, addressing modes, and registers.	The implementation of the architecture is called organization.
For designing a computer, its architecture is fixed first.	For designing a computer, an organization is decided after its architecture.
Computer Architecture is also called Instruction Set Architecture (ISA).	Computer Organization is frequently called microarchitecture.
Computer Architecture comprises logical functions such as instruction sets, registers, data types, and addressing modes.	Computer Organization consists of physical units like circuit designs, peripherals, and adders.
The different architectural categories found in our computer systems are as follows: 1. Von-Neumann Architecture 2. Harvard Architecture 3. Instruction Set Architecture 4. Micro-architecture 5. System Design	CPU organization is classified into three categories based on the number of address fields: 1. Organization of a single Accumulator. 2. Organization of general registers 3. Stack organization
It makes the computer's hardware visible.	It offers details on how well the computer performs.

Architecture coordinates the hardware and software of the system.	Computer Organization handles the segments of the network in a system.
The software developer is aware of it.	It escapes the software programmer's detection.
Examples- Intel and AMD created the x86 processor. Sun Microsystems and others created the SPARC processor. Apple, IBM, and Motorola created the PowerPC.	Organizational qualities include hardware elements that are invisible to the programmer, such as interfacing of computer and peripherals, memory technologies, and control signals.

UNIT – 01 – Part - B

REGISTER TRANSFER LANGUAGE AND MICRO OPERATIONS

Basic Definitions:

- ✓ A digital system is an interconnection of digital hardware modules.
- ✓ The modules are registers, decoders, arithmetic elements, and control logic.
- ✓ The various modules are interconnected with common data and control paths to form a digital computer system.
- ✓ Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them.
- ✓ The operations executed on data stored in registers are called *micro-operations*.
- ✓ A *micro-operation* is an elementary operation performed on the information stored in one or more registers.
- ✓ The result of the operation may replace the previous binary information of a register or may be transferred to another register.
- ✓ Examples of micro-operations are shift, count, clear, and load.
- ✓ The internal hardware organization of a digital computer is best defined by specifying:

    1. The set of registers it contains and their function.
    2. The sequence of micro-operations performed on the binary information stored in the registers.
    3. The control that initiates the sequence of micro-operations.

Register Transfer Language:

- ❖ The symbolic notation used to describe the micro-operation transfer among registers is called RTL (Register Transfer Language).
- ❖ The use of ***symbols*** instead of a ***narrative explanation*** provides an organized and concise manner for listing the micro-operation sequences in registers and the control functions that initiate them.
- ❖ A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.
- ❖ It is a convenient tool for describing the internal organization of digital computers in concise and

precise manner.

Registers:

- ❖ Computer registers are designated by upper case letters (and optionally followed by digits or letters) to denote the function of the register.
- ❖ For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name **MAR**.
- ❖ Other designations for registers are ***PC*** (for program counter), ***IR*** (for instruction register, and ***R1***
- ❖ (for processor register).
- ❖ The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1, starting from 0 in the rightmost position and increasing the numbers toward the left.

Figure 4-1 Block diagram of register.

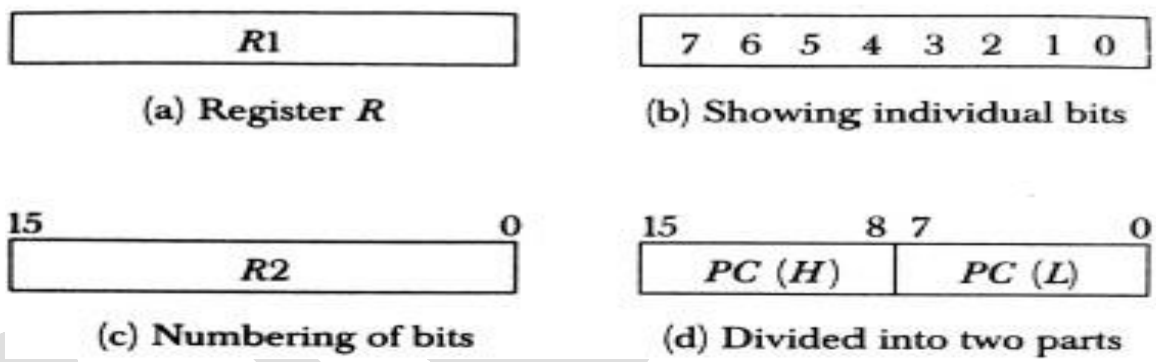


Figure shows the representation of registers in block diagram form.

- ❖ The most common way to represent a register is by a rectangular box with the name of the register inside, as in Fig.(a).
- ❖ The individual bits can be distinguished as in (b).
- ❖ The numbering of bits in a 16-bit register can be marked on top of the box as shown in (c).
- ❖ 16-bit register is partitioned into two parts in (d). Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol *H* (for high byte).
- ❖ The name of the 16-bit register is *PC*. The symbol *PC (0-7)* or *PC (L)* refers to the low-order byteand PC (8-15) or *PC (H)* to the high-order byte.

Register Transfer:

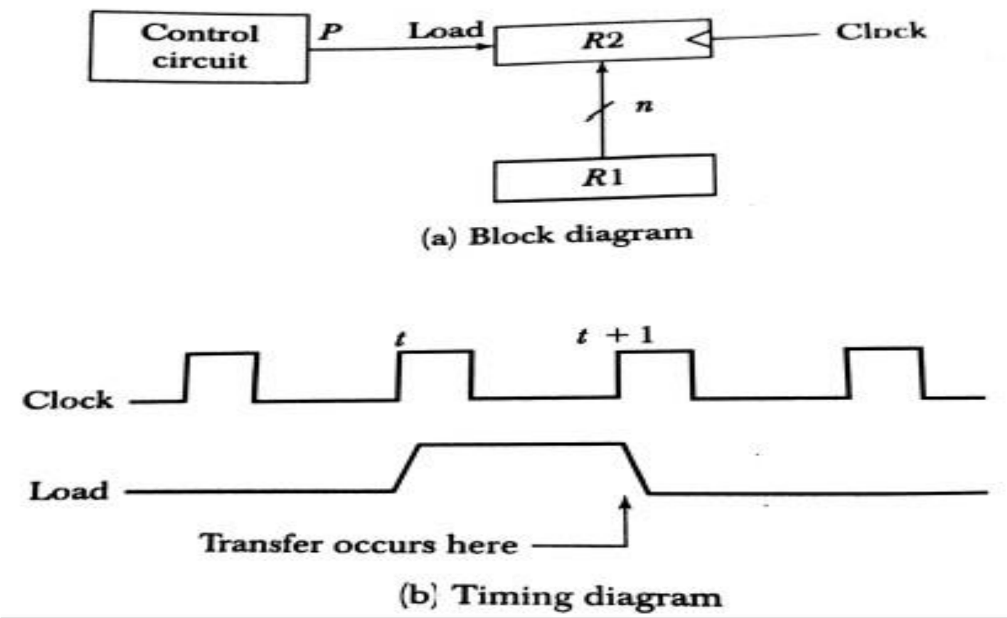
- Information transfer from one register to another is designated in symbolic form by means of a *replacement operator*.
- The statement **R2← R1** denotes a transfer of the content of register R1 into register R2.
- It designates a replacement of the content of R2 by the content of R1.
- By definition, the content of the source register R 1 does not change after the transfer.
- If we want the transfer to occur only under a predetermined control condition then it can be shown by an if-then statement.

  if (P=1) then R2← R1
- P is the control signal generated by a control section.
- We can separate the control variables from the register transfer operation by specifying a ***Control Function.***
- Control function is a Boolean variable that is equal to 0 or 1.
- control function is included in the statement as

  P: R2← R1
- Control condition is terminated by a colon implies transfer operation be executed by the hardware only if P=1.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.

Figure shows the block diagram that depicts the transfer from R1 to R2.

Figure 4-2 Transfer from R1 to R2 when $p = 1$.



- The n outputs of register R1 are connected to the n inputs of register R2.
- The letter n will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known.
- Register R2 has a load input that is activated by the control variable P.
- It is assumed that the control variable is synchronized with the same clock as the one applied to the register.
- As shown in the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time *t*.
- The next positive transition of the clock at time $t + 1$ finds the load input active and the data inputs of R2 are then loaded into the register in parallel.
- P may go back to 0 at time $t+1$; otherwise, the transfer will occur with every clock pulse transition while P remains active.
- Even though the control condition such as P becomes active just after time *t*, the actual transfer does not occur until the register is triggered by the next positive transition of the clock at time $t + 1$.
- The basic symbols of the register transfer notation are listed in below table

Symbol	Description	Examples
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow <--	Denotes transfer of information	R2 <-- R1
Comma ,	Separates two microoperations	R2 <-- R1, R1 <-- R2

A comma is used to separate two or more operations that are executed at the same time.

The statement

T : R2← R1, R1← R2 (exchange operation)

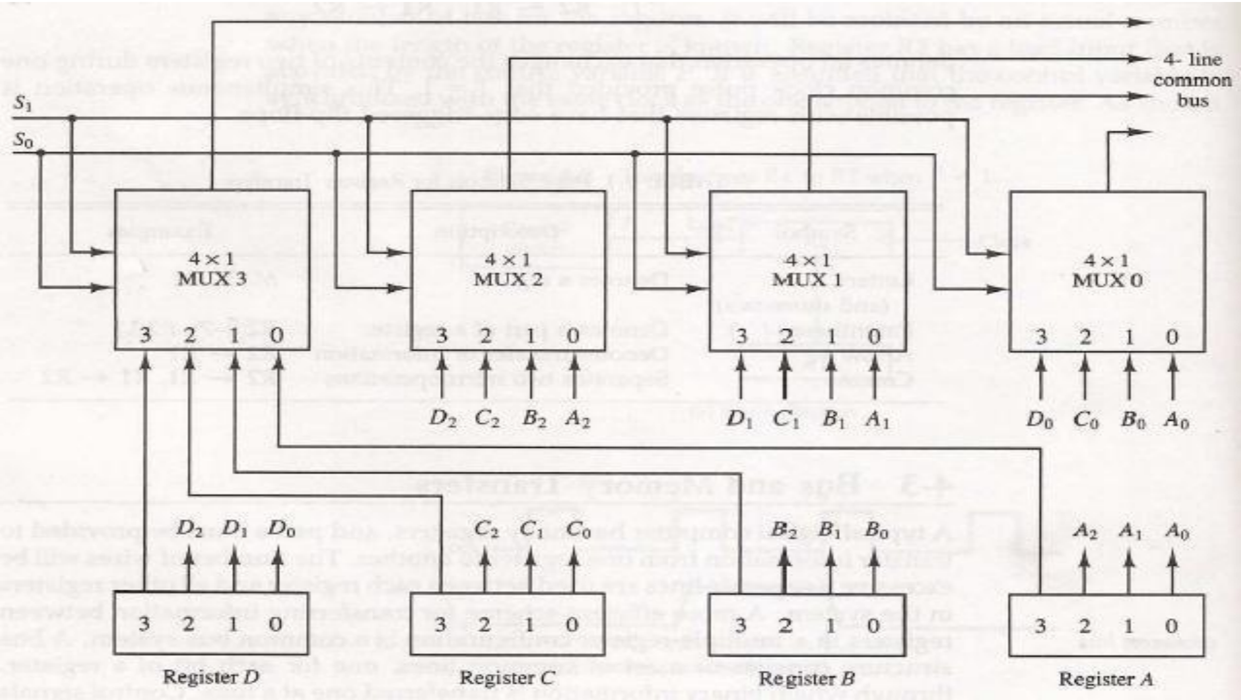
denotes an operation that exchanges the contents of two rgisters during one common clock pulse provided that T=1.

Bus and Memory Transfers:

- A more efficient scheme for transferring information between registers in ***a multiple-registerconfiguration*** is a ***Common Bus System.***
- A common bus consists of a set of common lines, one for each bit of a register.
- Control signals determine which register is selected by the bus during each particular registertransfer.
- Different ways of constructing a Common Bus System
  - ✓ Using Multiplexers
  - ✓ Using Tri-state Buffers

Common bus system is with multiplexers:

- The multiplexers select the source register whose binary information is then placedon the bus.
- The construction of a bus system for four registers is shown in below Figure.



- The bus consists of four 4 x 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S_1 and S_0.
- For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labelled A_1.
- The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus.
- Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of theregisters, and similarly for the other two bits.
- The two selection lines Si and So 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 S_1S_0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus.
- This 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, register *B* is selected if S_1S_0 = 01, and so on.
- Table shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

S_1	S_0	Register selected
0	0	*A*
0	1	*B*
1	0	*C*
1	1	*D*

- In general a bus system has
  - ✓ multiplex "k" Registers
  - ✓ each register of "n" bits
  - ✓ to produce "n-line bus"
  - ✓ no. of multiplexers required = n
  - ✓ size of each multiplexer = k x 1
- When the bus is includes in the statement, the register transfer is symbolized as follows:

BUS← C, R1← BUS

- The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating its load control input. If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

R1← C

Arithmetic Micro-operations:

- The basic arithmetic micro-operations are
  - ✓ Addition
  - ✓ Subtraction
  - ✓ Increment
  - ✓ Decrement
  - ✓ Shift
- The arithmetic Micro-operation defined by the statement below specifies the add micro-operation.

R3 ← R1 + R2

- It states that the contents of R1 are added to contents of R2 and sum is transferred to R3.
- To implement this statement hardware requires 3 registers and digital component that performs addition
- Subtraction is most often implemented through complementation and addition.
- The subtract operation is specified by the following statement

$$R3 \leftarrow R1 + \overline{R2} + 1$$

- instead of minus operator, we can write as
- $\overline{R2}$ is the symbol for the 1's complement of R2
- Adding 1 to 1's complement produces 2's complement
- Adding the contents of *R1* to the 2's complement of R2 is equivalent to R1-R2.

Binary Adder:

- Digital circuit that forms the arithmetic sum of 2 bits and the previous carry is called **FULL ADDER.**
- Digital circuit that generates the arithmetic sum of 2 binary numbers of any lengths is called
- Figure shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.

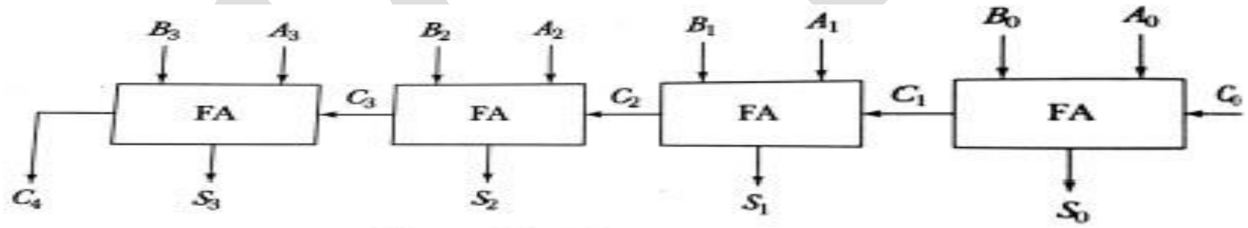


Figure 4-6 4-bit binary adder.

- The augends bits of A and the addend bits of *B* are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders. The input carry to the binary adder is Co and the output carry is C4. The *S* outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders.

Binary Adder – Subtractor:

- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.

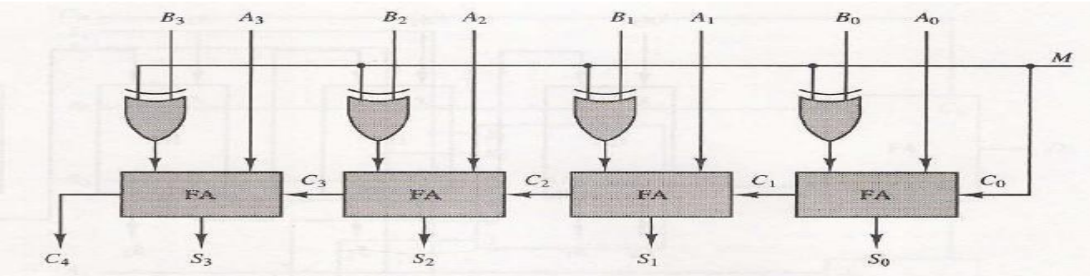


Figure 4-7 4-bit adder-subtractor.

- A 4-bit adder-subtractor circuit is shown in Fig..
- The mode input M controls the operation. When M = 0 the circuit is an adder and when M = 1 the circuit becomes a subtractor.
- Each exclusive-OR gate receives input M and one of the inputs of B
- When M = 0, we have *B* xor 0 = B. The full-adders receive the value of *B*, the input carry is 0, andthe circuit performs A plus B.
- When M = 1, we have *B xor 1* = *B'* and Co = 1.
- The *B* inputs are all complemented and a 1 is added through the input carry.
- The circuit performs the operation A plus the 2's complement of *B*.

Binary Incrementer:

- The increment microoperation adds one to a number in a register.
- For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.
- This can be accomplished by means of half-adders connected in cascade.
- The diagram of a 4-bit 'combinational circuit incrementer is shown in Fig..
- One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.
- The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.

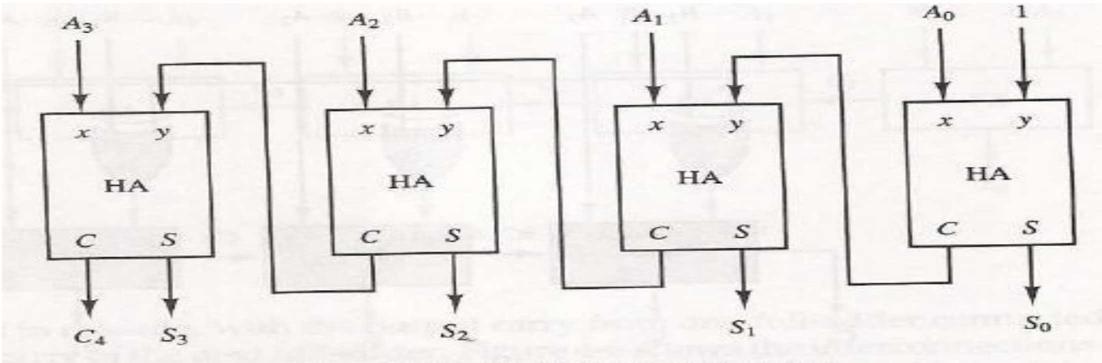
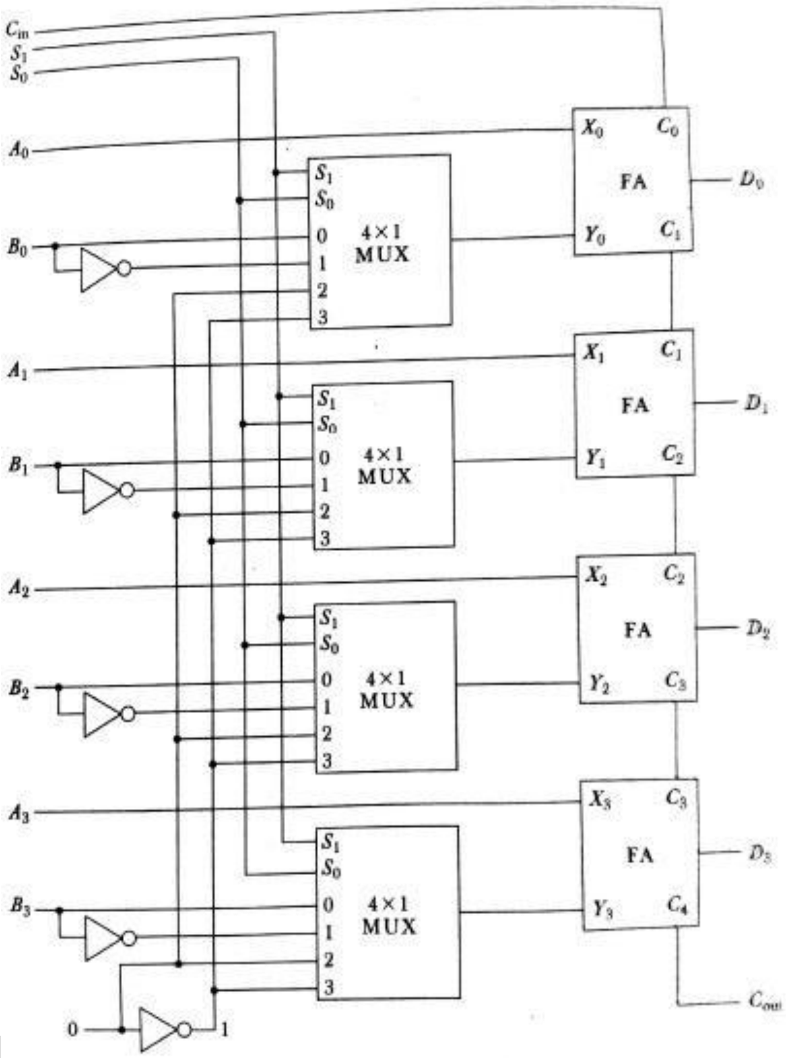


Figure 4-8 4-bit binary incrementer.

- The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3.
- The output carry C_4 will be 1 only after incrementing binary 1111. This also causes outputs S_0 through S_3 to go to 0.
- The circuit of Fig. can be extended to an *n* -bit binary incrementer by extending the diagram to include *n* half-adders.
- The least significant bit must have one input connected to logic-1. The other inputs receive the number to be incremented or the carry from the previous stage.

Arithmetic Circuit:

- The basic component of an arithmetic circuit is the parallel adder.
- By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.
- The diagram of a 4-bit arithmetic circuit is shown in Fig.. It has four full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations.



- There are two 4-bit inputs A and *B* and a 4-bit output *D*.
- The four inputs from A go directly to the X inputs of the binary adder.
- Each of the four inputs from B are connected to the data inputs of the multiplexers.
- The multiplexers data inputs also receive the complement of B.
- The other two data inputs are connected to logic-0 and logic-1.
- The four multiplexers are controlled by two selection inputs S_1 and S_0. The input carry C_{in}, goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- By controlling the value of Y with the two selection inputs S_1 and S_0 and making C_{in} equal to 0 or 1,it is possible to generate the eight arithmetic microoperations listed in Table .

TABLE 4-4 Arithmetic Circuit Function Table

Select			Input	Output	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Addition:

- When S_1S_0= 00, the value of *B* is applied to the Y inputs of the adder.
  - ✓ If Cir, = 0, the output *D =A+B.*
  - ✓ If Cin = 1, output *D=A+B + 1.*
- Both cases perform the add microoperation with or without adding the input carry.

Subtraction:

- When S_1S_0 = 01, the complement of B is applied to the Y inputs of the adder.
  - ✓ If C_{in} = 1, then D = $\bar{A}$ + B + 1. This produces A plus the 2's complement of B, which isequivalent to a subtraction of A -B.
  - ✓ When C_{in} = 0 then D = A $\bar{+}$ B. This is equivalent to a subtract with borrow, that is, A-B-1.

Increment:

- When S_1S_0 = 10, the inputs from *B* are neglected, and instead, all 0's are inserted into the Y inputs. The output becomes D = A + 0 + C_{in}. This gives *D* = A when C_{in} = 0 and D = A + 1 when C_{in} = 1.
- In the first case we have a direct transfer from input A to output D.
- In the second case, the value of A is incremented by 1.

Decrement:

- When S_1S_0= 11, all l's are inserted into the Y inputs of the adder to produce the decrement operation $D = A - 1$ when $C_{in} = 0$.
- This is because a number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces F = A + 2's complement of 1 = A — 1. When $C_{in} = 1$, then $D = A - 1 + 1 = A$, which causes a direct transfer from input A to output D.

Logic Micro-operations:

- Logic microoperations specify binary operations for strings of bits stored in registers.
- These operations consider each bit of the register separately and treat them as binary variables.
- For example, the exclusive-OR microoperation with the contents of two registers RI and R2 is symbolized by the statement

$$P: \quad R1 \leftarrow R1 \oplus R2$$

- It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1.

List of Logic Microoperations:

- There are 16 different logic operations that can be performed with two binary variables.
- They can be determined from all possible truth tables obtained with two binary variables as shown in Table.

TABLE 4-5 Truth Tables for 16 Functions of Two Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

- The 16 Boolean functions of two variables x and y are expressed in algebraic form in the first column of Table.
- The 16 logic microoperations are derived from these functions by replacing variable x by the binary content of register A and variable y by the binary content of register B.
- The logic micro-operations listed in the second column represent a relationship between the binary content of two registers A and B.

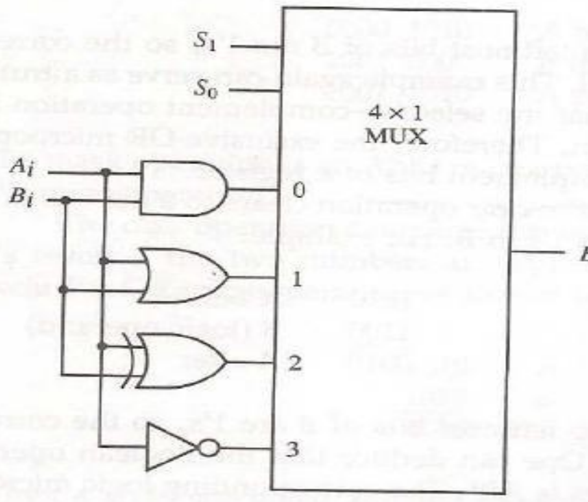
TABLE 4-6 Sixteen Logic Microoperations

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware Implementation:

- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
- Although there are 16 logic microoperations, most computers use only four--AND, OR, XOR (exclusive-OR), and complement from which all others can be derived.
- Figure shows one stage of a circuit that generates the four basic logic microoperations.
- It consists of four gates and a multiplexer. Each of the four logic operations is generated through a gate that performs the required logic.
- The outputs of the gates are applied to the data inputs of the multiplexer. The two selection inputs S_1 and S_0 choose one of the data inputs of the multiplexer and direct its value to the output.

Figure 4-10 One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

Shift Microoperations:

- ❖ Shift microoperations are used for serial transfer of data.
- ❖ The contents of a register can be shifted to the left or the right.
- ❖ During a shift-left operation the serial input transfers a bit into the rightmost position.
- ❖ During a shift-right operation the serial input transfers a bit into the leftmost position.
- ❖ There are three types of shifts: logical, circular, and arithmetic.
- ❖ The symbolic notation for the shift microoperations is shown in Table.

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

✓ **Logical Shift:**

- A *logical* shift is one that transfers 0 through the serial input.
- The symbols *shl* and shr for logical shift-left and shift-right microoperations.
- The microoperations that specify a 1-bit shift to the left of the content of register R and a 1-bit shift to the right of the content of register R shown in table.
- The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

✓ **Circular Shift:**

- The *circular* shift (also known as a *rotate* operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input.
- We will use the symbols *cil* and *cir* for the circular shift left and right, respectively.

✓ **Arithmetic Shift:**

- An *arithmetic shift* is a microoperation that shifts a signed binary number to the left or right.
- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.
- Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.

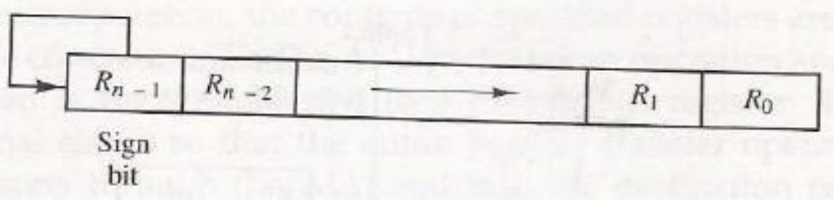
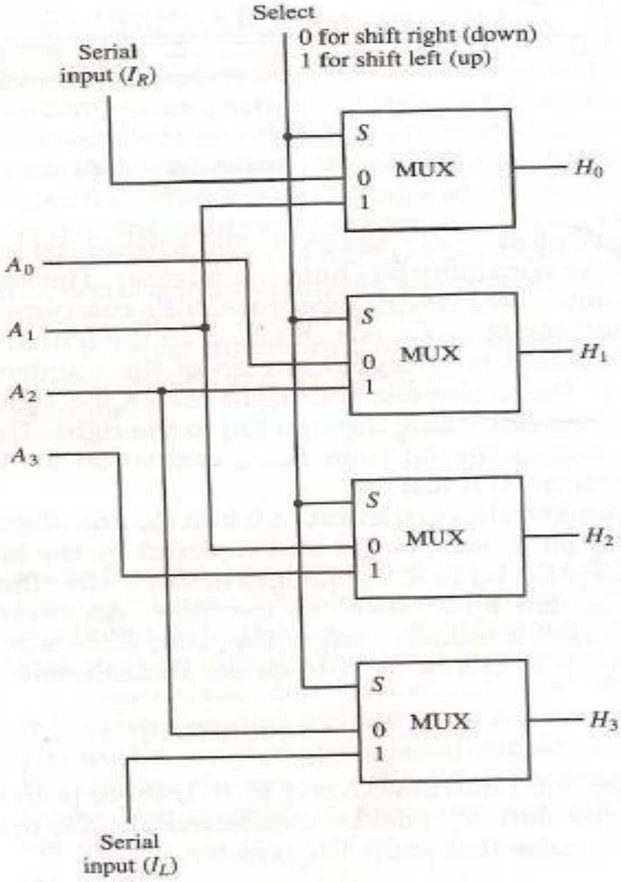


Figure 4-11 Arithmetic shift right.

Hardware Implementation:

- A combinational circuit shifter can be constructed with multiplexers as shown in Fig.
- The 4-bit shifter has four data inputs, A_0 through A_3, and four data outputs, H_0 through H_3.
- There are two serial inputs, one for shift left (I_L) and the other for shift right (I_R).
- When the selection input *S=0* the input data are shifted right (down in the diagram).

- ❖ When $S = 1$, the input data are shifted left (up in the diagram).
- ❖ The function table in Fig. shows which input goes to each output after the shift.
- ❖ A shifter with n data inputs and outputs requires n multiplexers.
- ❖ The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

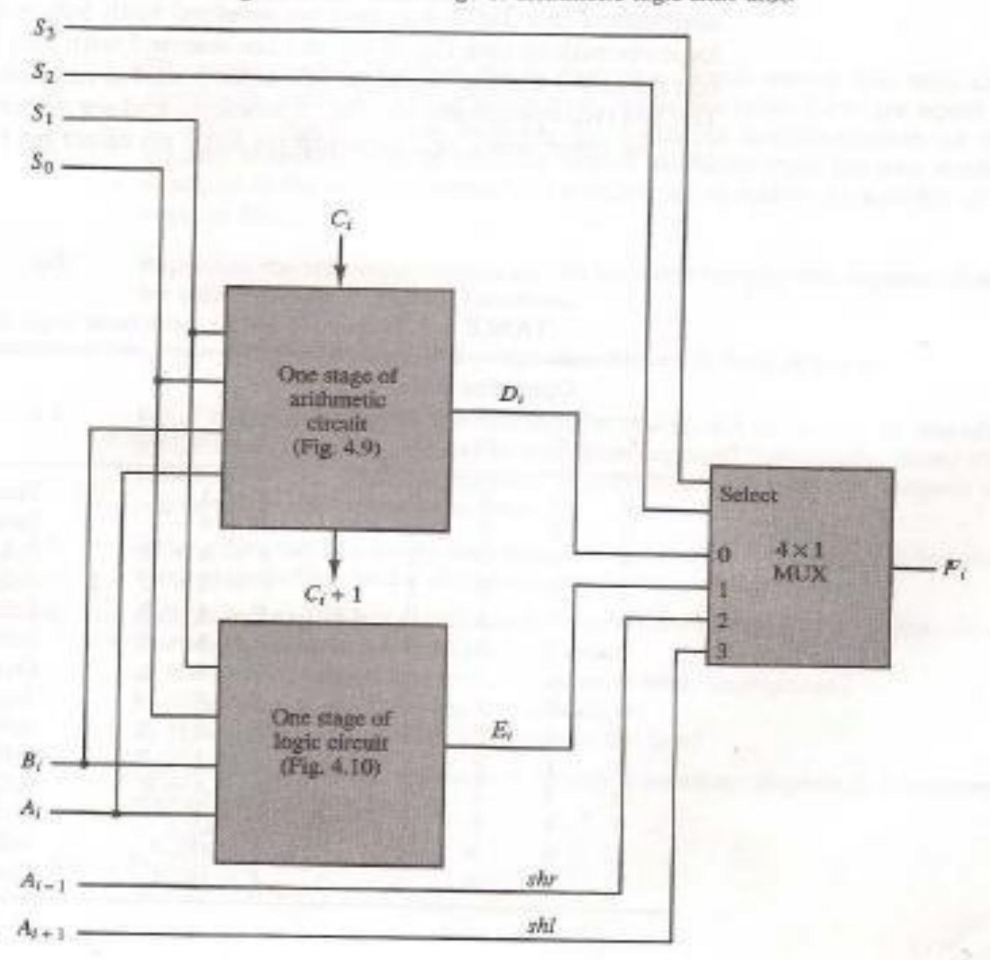
Figure 4-12 4-bit combinational circuit shifter.

Arithmetic Logic Shift Unit:

- ❖ Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.
- ❖ The ALU is a combinational circuit so that the entire register transfer operation from the
- ❖ source registers through the ALU and into the destination register can be performed during one clock pulse period.
- ❖ The shift microoperations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.

- The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables. One stage of an arithmetic logic shift unit is shown in Fig.
- Particular microoperation is selected with inputs S_1 and S_0. A 4 x 1 multiplexer at the output chooses between an arithmetic output in D_i and a logic output in E_i.
- The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift-right operation and A_{i+1} for the shift-left operation.
- The circuit whose one stage is specified in Fig. 4-13 provides eight arithmetic operation, four logic operations, and two shift operations.
- Each operation is selected with the five variables S_3, S_2, S_1, S_0 and C_{in}.
- The input carry C_{in} is used for selecting an arithmetic operation only.

Figure 4-13 One stage of arithmetic logic shift unit.



- Table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with $S_3S_2 = 00$.

- The next four are logic and are selected with $S_3S_2 = 01$.
- The input carry has no effect during the logic operations and is marked with don't-care x's.
- The last two operations are shift operations and are selected with $S_3S_2 = 10$ and 11.

TABLE 4-8 Function Table for Arithmetic Logic Shift Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
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	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F

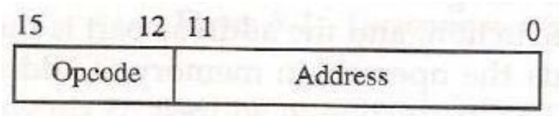
- The other three selection inputs have no effect on the shift.

UNIT – 01 – Part - C

BASIC COMPUTER ORGANIZATION AND DESIGN

1. InstructionCodes:

- Theorganizationofthecomputerisdefinedbyitsinternalregisters,thetimingandcontrolstructure, and the set of instructions that ituses.
- Internal organization of a computer is defined by the sequence of micro-operations it performs on data stored in its registers.
- Computer can be instructed about the specific sequence of operations it mustperform.
- User controls this process by means of aProgram.
- ***Program:***setofinstructionsthatspecifytheoperations,operands,andthesequencebywhich processing has tooccur.
- ***Instruction****:* a binary code that specifies a sequence of micro-operations for thecomputer.
- The computer reads each instruction from memory and places it in a control register. The control theninterpretsthebinarycodeoftheinstructionandproceedstoexecuteitbyissuingasequenceof micro-operations. – *InstructionCycle*
- **Instruction Code:** group of bits that instruct the computer to perform specificoperation.
- Instruction code is usually divided into two parts: Opcode andaddress(operand)



Instruction format

Operation Code(opcode):

- ✓ Group of bits that define theoperation
- ✓ Eg: add, subtract, multiply, shift,complement.
- ✓ No. of bits required for opcode depends on no. of operations available incomputer.
- ✓ n bit opcode>= 2^n (or less)operations

Address(operand):

- ✓ Specifies the location of operands (registers or memorywords)
- ✓ Memory words are specified by their address
- ✓ Registers are specified by their k-bit binarycode

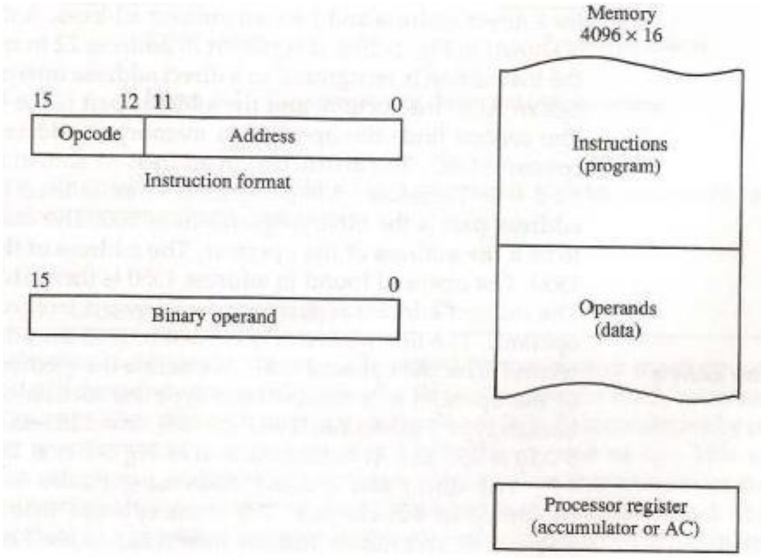
- ✓ k-bit address >= 2^k registers

Stored Program Organization:

The ability to store and execute instructions is the most important property of a general-purpose computer. That type of stored program concept is called stored programorganization.

The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies anaddress.

The below figure shows the stored programorganization



- ➢ Instructions are stored in one section of memory and data inanother.
- ➢ For a memory unit with 4096 words we need 12 bits to specify an address since 2^{12} =4096.
- ➢ If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of anoperand.
- ➢ ***Accumulator(AC):***
  - ✓ Computers that have a single-processor register usually assign to it the name accumulator and label itAC.
  - ✓ The operation is performed with the memory operand and the content ofAC.

Addressing of Operand:

Sometimes convenient to use the address bits of an instruction code not as an address but as the actualoperand.

When the second part of an instruction code specifies an operand, the instruction is said to havean

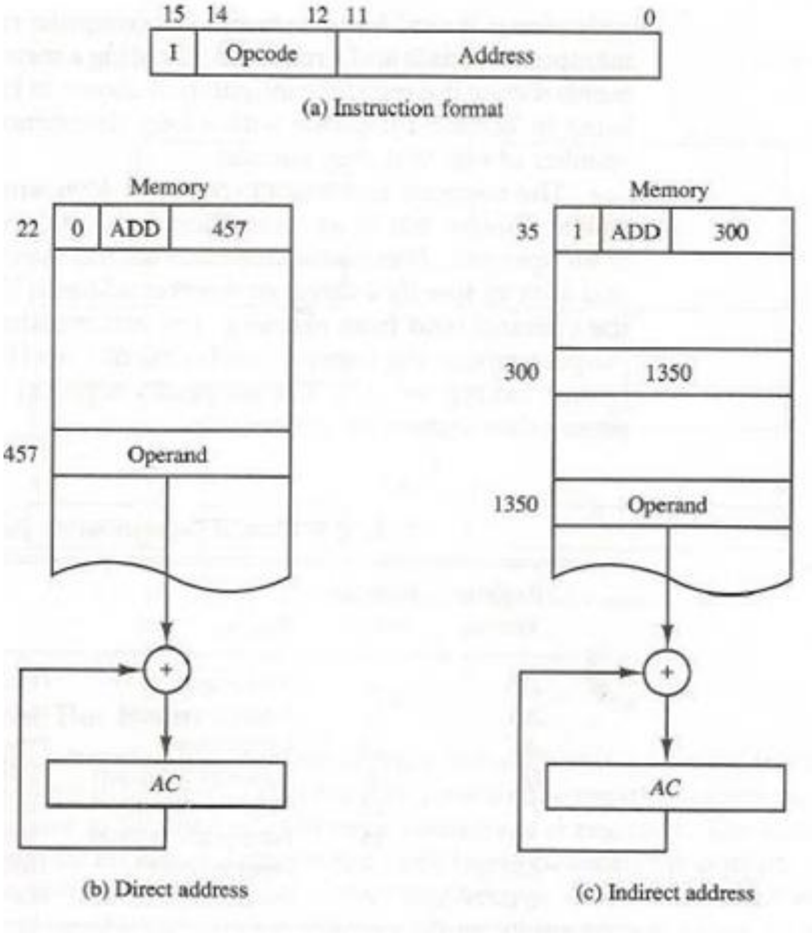
immediate operand.

When the second part specifies the address of an operand, the instruction is said to have a ***direct address.***

Whensecondpartoftheinstructiondesignateanaddressofamemoryword inwhichtheaddressof the operand is found such instruction have ***indirectaddress.***

One bit of the instruction code can be used to distinguish between a direct and an indirectaddress.

The instruction code format shown in Fig. 5-2(a). It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. The mode bit is 0 for a direct address and 1 for an indirectaddress.



It is placed in address 22 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.

The control finds the operand in memory at address 457 and adds it to the content ofAC.

The instruction in address 35 shown in Fig.(c) has a mode bit I =1.Therefore, it is recognized as an indirect addressinstruction.

Theaddresspartisthebinaryequivalentof300.Thecontrolgoestoaddress300tofindtheaddress of the operand. The address of the operand in this case is1350.The operand found in address 1350 is then added to the content of*AC*.

The ***effective address*** to be the address of the operand in a computation-type instruction or the target address in a branch-typeinstruction.Thus the effective address in the instruction of Fig.(b) is 457 and in the instruction of Fig.(c) is 1350.

2. Computer Registers:

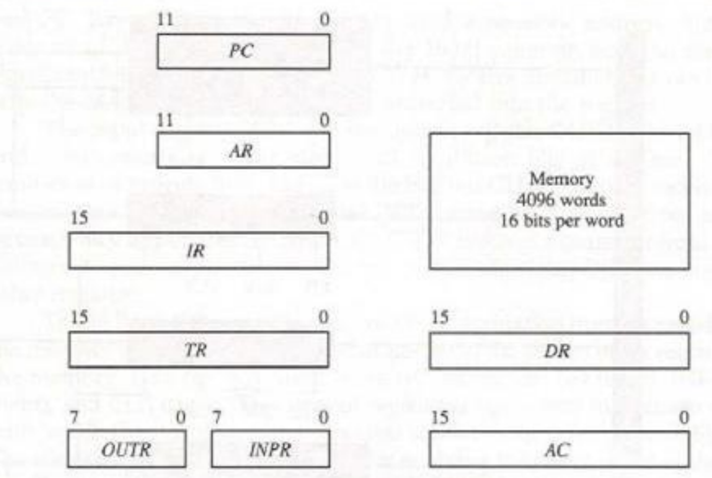
What is the need for computerregisters?

The need of the registers in computerfor;

- ❖ Instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed**(PC)**.
- ❖ Necessarytoprovidearegisterinthecontrolunitforstoringtheinstructioncodeafter it is read from memory**(IR)**.
- ❖ Needs processor registers for manipulating data (AC and TR) and a register for holding a memory address**(AR)**.

List of Registers for the Basic Computer

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



Basic computer registers and memory.

- ✓ The *data register (DR)* holds the operand read frommemory.
- ✓ The *accumulator (AC)* register is a general purpose processingregister.
- ✓ The instruction read from memory is placed in the *instruction register(IR)*.
- ✓ The *temporary register (TR)* is used for holding temporary data during theprocessing.
- ✓ The *memory address register (AR)* has 12 bits since this is the width of a memoryaddress.
- ✓ The *program counter (PC)* also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction isexecuted.
- ✓ Two registers are used for input andoutput.
  - ➢ The *input register (INPR)* receives an 8-bit character from an inputdevice.
  - ➢ The *output register (OUTR)* holds an 8-bit character for an outputdevice.

Common Bus System:

The basic computer has eight registers, a memory unit, and a controlunit. Paths must be provided to transfer information from one register to another and betweenmemory andregisters.A more efficient scheme for transferring information in a system with many registers is to use a commonbus.

The connection of the registers and memory of the basic computer to a common bus system is shown in Fig.

The outputs of seven registers and memory are connected to the commonbus.

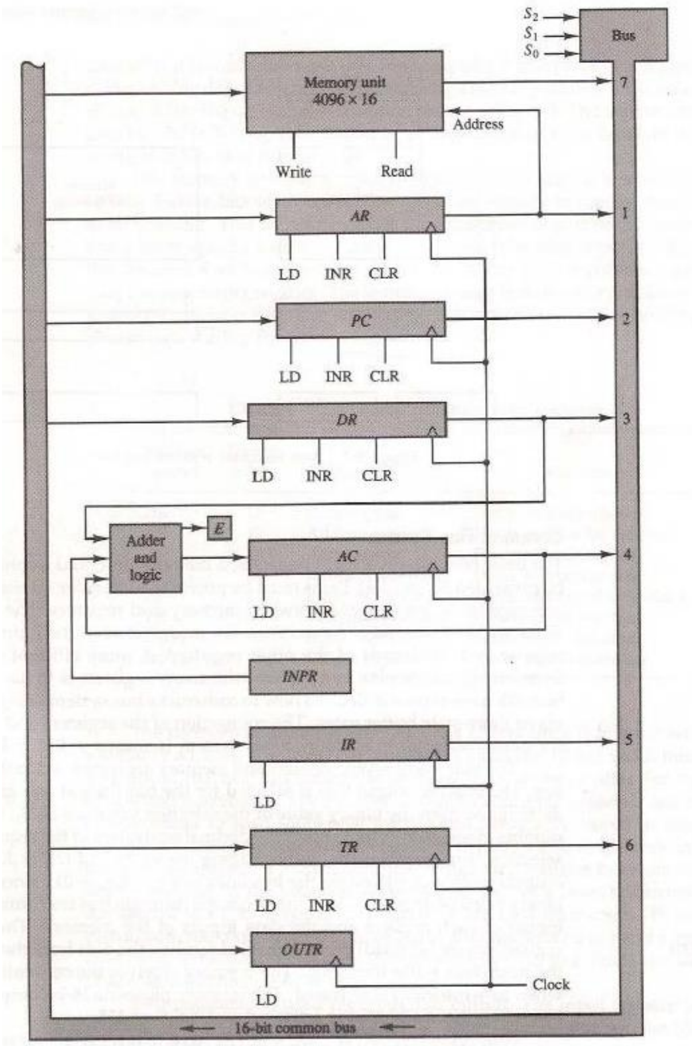
Thespecificoutputthatisselectedforthebuslinesatanygiventimeisdeterminedfromthebinary value of the selection variables S_2, S_1, andS_0.

The number along each output shows the decimal equivalent of the required binaryselection.

For example, the number along the output of *DR* is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0$ = 011.

Thelinesfromthecommonbusareconnectedtotheinputsofeachregisterandthedatainputsof thememory.

- ➢ The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulsetransition.
- ➢ The memory receives the contents of the bus when its write input isactivated.
- ➢ The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0$ =111.
- ➢ Two registers, AR and *PC*, have 12 bits each since they hold a memoryaddress.
- ➢ Whenthecontentsof*AR*or*PC*areappliedtothe16-bitcommonbus,thefourmostsignificantbits are set to 0's.



Basic computer registers connected to a common bus.

WhenARor*PC*receivesinformationfromthebus,onlythe12leastsignificantbitsaretransferred into theregister.

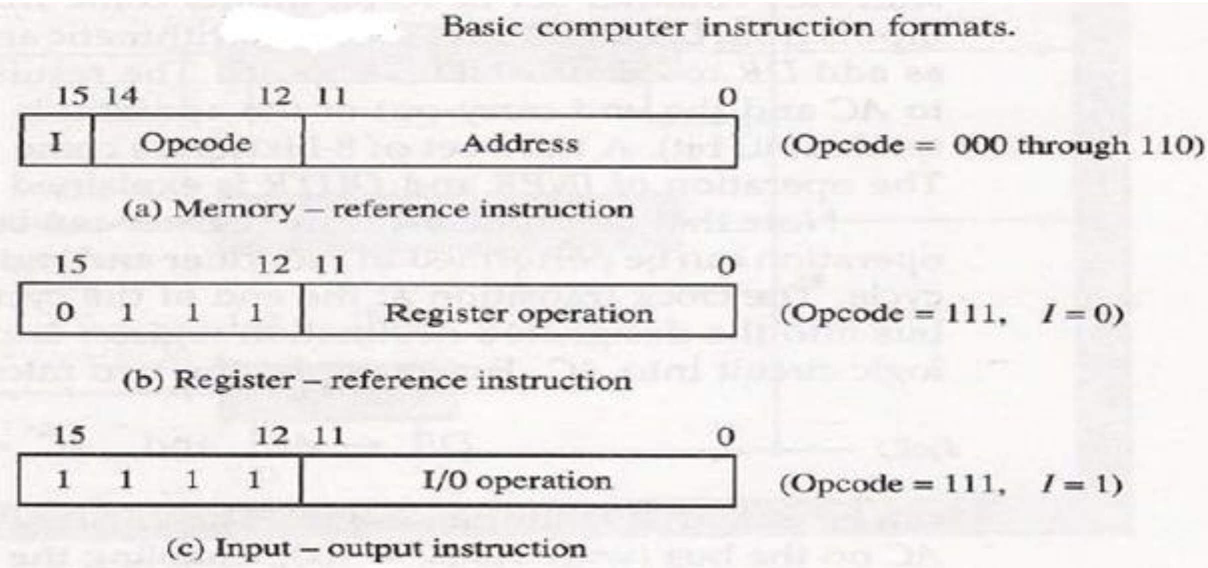
- The input register *INPR* and the output register OUTR have 8 bitseach.
- They communicate with the eight least significant bits in thebus.
- INPRisconnectedtoprovideinformationtothebusbutOUTRcanonlyreceiveinformationfromthe bus.
- This is because INPR receives a character from an input device which is then transferred to*AC*.
- *OUTR* receives a character from AC and delivers it to an output device.
- Five registers have three control inputs: LD (load), INR (increment), and CLR(clear).
- This type of register is equivalent to a binary counter with parallel load and synchronousclear.
- Two registers have only a LDinput.
- Theinputdataandoutputdataofthememoryareconnectedtothecommonbus,butthememory address is

connected to AR.

- Therefore, AR must always be used to specify a memoryaddress.
- The 16 inputs of *AC* come from an adder and logic circuit. This circuit has three sets ofinputs.
  - ✓ One set of 16-bit inputs come from the outputs of*AC*.
  - ✓ Another set of 16-bit inputs come from the data register*DR*.
  - ✓ The result of an addition is transferred to *AC* and the end carry-out of the addition is transferred to flip-flop E (extended *AC*bit).
  - ✓ A third set of 8-bit inputs come from the input registerINPR.
- The content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clockcycle.
- For example, the two microoperations DR□AC and AC □ DR can be executed at the sametime.
- This can be done by placing the content of *AC* on the bus (with $S_2S_1S_0$ = 100), enabling the LD (load) input of DR, transferring the content of DR through the adder and logic circuit into AC, and enabling the LD (load) input of *AC*, all during the same clockcycle.

3. Computer Instructions:

The basic computer has three instruction code formats, as shown in Fig.. Each format has 16bits.



Theoperationcode(opcode)partoftheinstructioncontainsthreebitsandthemeaningofthe remaining 13 bits depends on the operation codeencountered.

A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I.

I is equal to 0 for direct address and to 1 for indirectaddress.

Theregister-referenceinstructionsarerecognizedbytheoperationcode1.11witha0intheleftmost bit (bit 15) of theinstruction.

A register-reference instruction specifies an operation on the AC register. So an operand from memoryisnotneeded.Therefore,theother12bitsareusedtospecifytheoperationtobeexecuted.

Aninput—outputinstructiondoesnotneedareferencetomemoryandisrecognizedbythe operation code 111 with a 1 in the leftmost bit of theinstruction.

The remaining 12 bits are used to specify the type of input—outputoperation.

The instructions for the computer are listed in Table.

Basic Computer Instructions

	Hexadecimal code		
Symbol	$I = 0$	$I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

The symbol designation is a three-letter word and represents an abbreviation intended for programmers andusers.

The hexadecimal code is equal to the equivalent hexadecimal number of the binary code used for the instruction.

Instruction Set Completeness:

- Acomputershouldhaveasetofinstructionssothattheusercanconstructmachinelanguage programs to evaluate anyfunction.
- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the followingcategories:
  - ✓ Arithmetic, logical, and shiftinstructions
  - ✓ Data Instructions (for moving information to and from memory and processorregisters)
  - ✓ Program control orBrach
  - ✓ Input and outputinstructions
- There is one arithmetic instruction, ADD, and two related instructions, complement AC(CMA)and increment AC(INC). With these three instructions we can add and subtract binary numbers when negative numbers are in signed-2's complementrepresentation.
- Thecirculateinstructions,CIRandCIL;canbeusedforarithmeticshiftsaswellasanyother type of shiftsdesired.
- There are three logic operations: AND, complement AC (CMA), and clear AC(CLA). The ANDand complement provide a NANDoperation.
- Moving information from memory to *AC* is accomplished with the load AC (LDA) instruction. Storing information from AC into memory is done with the store AC (STA)instruction.
- The branch instructions BUN, BSA, and ISZ, together with the four skip instructions, provide capabilities for program control and checking of statusconditions.
- Theinput(INP}andoutput(OUT)instructionscauseinformationtobetransferredbetweenthe computer and externaldevices.

4. Timing andControl:

The timing for all registers in the basic computer is controlled by a master clockgenerator.The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the controlunit.

The clock pulses do not change the state of a register unless the register is enabled by a control signal.The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for theaccumulator.

There are two major types of controlorganization:

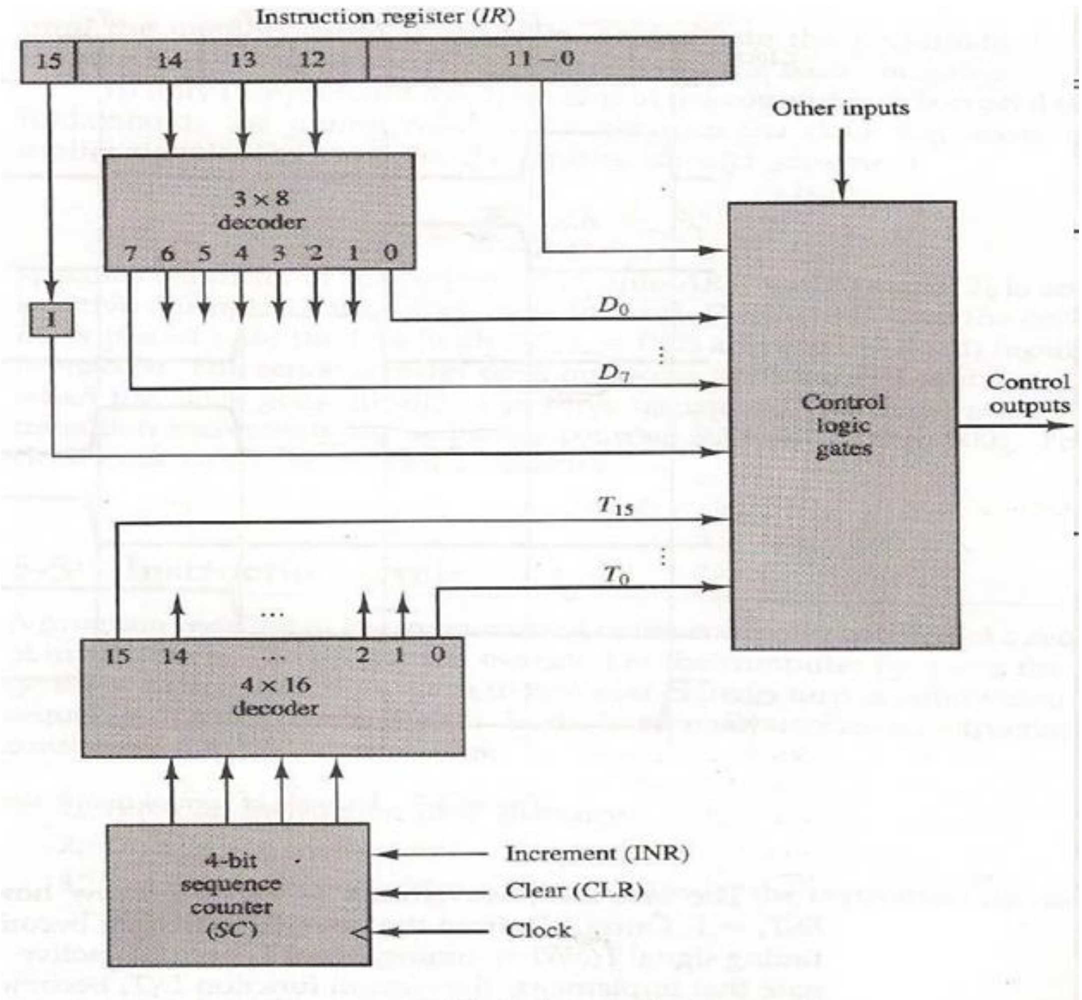
- *Hardwiredcontrol*
- *Microprogrammedcontrol*

The differences between hardwired and microprogrammed controlare

Hardwired Control	Microprogrammed Control
✓ The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.	✓ The control information is stored in a control memory. The control memoryis programmed to initiate therequired sequence of microoperations.
✓ The advantage that it can be optimizedto produce a fast mode of operation.	✓ Compared with the hardwiredcontrol operation is slow.
✓ Requires changes in the wiring among the various components if the design has tobe modified or changed.	✓ Required changes or modifications can be done by updating the microprogramin controlmemory.

The block diagram of the hardwired control unit is shown in Fig.It consists of two decoders, a sequence counter, and a number of control logicgates.

An instruction read from memory is placed in the instruction register (IR). It is divided into three parts: The I bit, the operation code, and bits 0 through11.The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 throughD_7.Bit 15 of the instruction is transferred to a flip-flop designated by the symbolI.Bits 0 through 11 are applied to the control logicgates.The 4-bit sequence counter can count in binary from 0 through15.



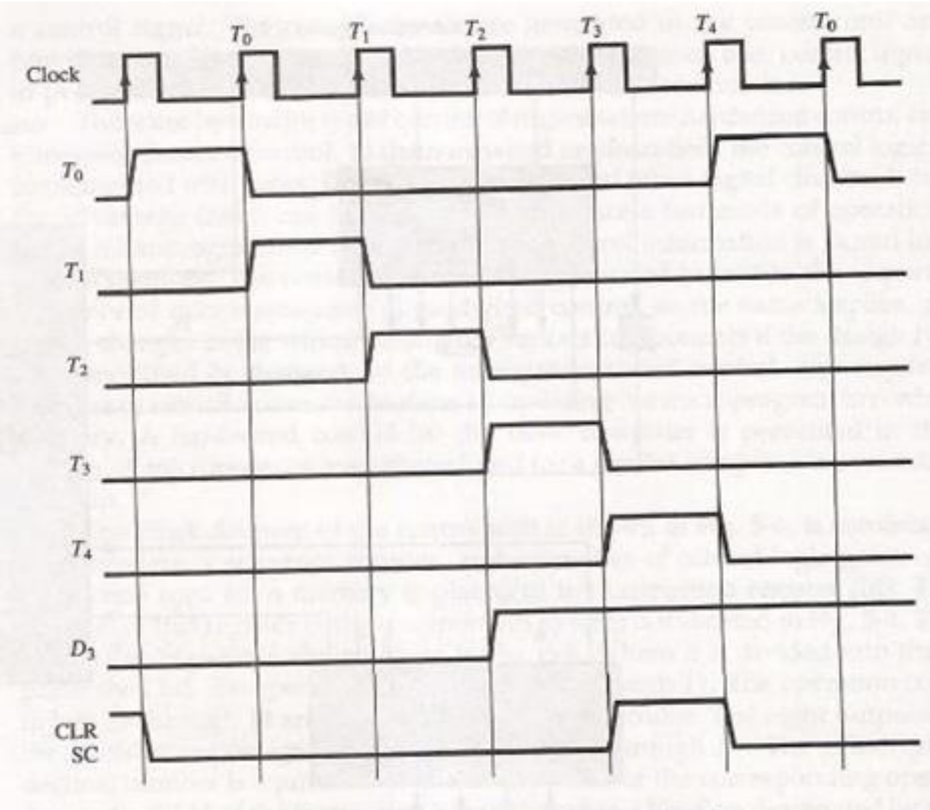
Control unit of basic computer.

- ❖ The outputs of the counter are decoded into 16 timing signals T_0throughtT_{15}.
- ❖ The sequence counter *SC* can be incremented or clearedsynchronously.
- ❖ The counter is incremented to provide the sequence of timing signals out of the 4 x 16decoder.
- ❖ As an example, consider the case where *SC* is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence. At time T_4, *SC* is cleared to 0 if decoder output D3 isactive.
- ❖ This is expressed symbolically by thestatement

D_3T_4: SC□0

- ❖ The timing diagram of Fig. shows the time relationship of the controlsignals.
- ❖ The sequence counter *SC* responds to the positive transition of theclock.
- ❖ Initially, the CLR input of *SC* is active. The first positive transition of the clock clears *SC* to 0, which in turn activates the timing signal T_0 out of the decoder. T_0 is active during one clockcycle.
- ❖ SC is incremented with every positive clock transition, unless its CLR input isactive.
- ❖ This produces the sequence of timing signals T_0, T_1, T_2, T_3, $T_4$and so on, as shown in thediagram.

- The last three waveforms in Fig. show how *SC* is cleared when D_3T_4 =1.
- Output D_3 from the operation decoder becomes active at the end of timing signalT_2.
- WhentimingsignalT_4becomesactive,theoutputoftheANDgatethatimplementsthecontrol function D_3T_4 becomesactive.
- This signal is applied to the CLR input of *SC*. On the next positive clock transition (the one marked T4 in the diagram) the counter is cleared to0.
- This causes the timing signal T_0 to become active instead of T_5 that would have been active if *SC* were incremented instead ofcleared.



Example of control timing signals.

5. Instruction Cycle:

A program residing in the memory unit of the computer consists of a sequence ofinstructions.The program is executed in the computer by going through a cycle for eachinstruction.

Each instruction cycle in turn is subdivided into a sequence of sub cycles or phases.

In the basic computer each instruction cycle consists of the followingphases:

1. Fetch an instruction frommemory.
2. Decode theinstruction.

3. Read the effective address from memory if the instruction has an indirectaddress.
4. Execute theinstruction.

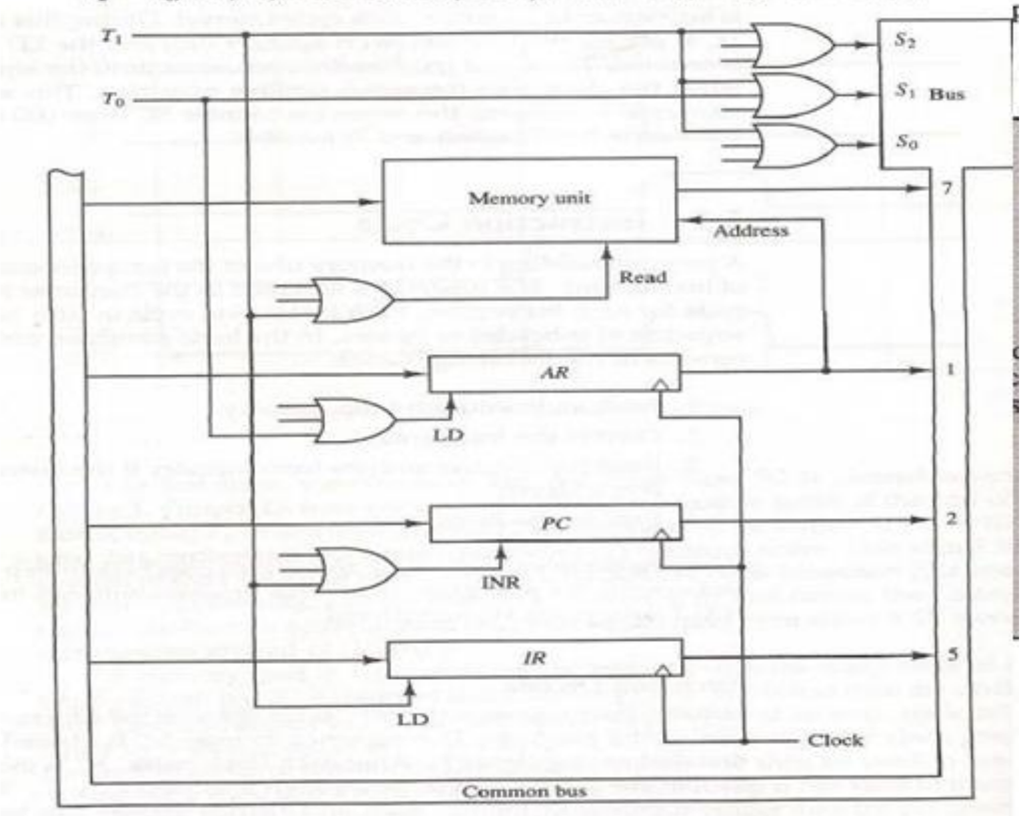
Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the nextinstruction.

Fetch and Decode:

Initially, the program counter PC is loaded with the address of the first instruction in theprogram.The sequence counter *SC* is cleared to 0, providing a decoded timing signalT_0.

The microoperations for the fetch and decode phases can be specified by the followingregister transfer statements.

$$T_0: \quad AR \leftarrow PC$$
$$T_1: \quad IR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$
$$T_2: \quad D_0, \ldots, D_7 \leftarrow \text{Decode } IR(12\text{–}14), \quad AR \leftarrow IR(0\text{–}11), \quad I \leftarrow IR(15)$$



Register transfers for the fetch phase.

Above Figure shows how the first two register transfer statements are implemented in the bussystem.To provide the data path for the transfer of *PC* to AR we must apply timing signal T_0 to achieve the followingconnection:

- ✓ Placethecontent of*PC*ontothebusbymakingthebusselectioninputsS_2,S_1,S_0equalto010.
- ✓ Transfer the content of the bus to AR by enabling the LD input ofAR.

In order to implement the second statement it is necessary to use timing signal $T_1$to provide the following connections in the bussystem.

- ✓ Enable the read input ofmemory.
- ✓ Place the content of memory onto the bus by making$S_2S_1S_0$=111.
- ✓ Transfer the content of the bus to IR by enabling the LD input ofIR.
- ✓ Increment PC by enabling the INR input ofPC.

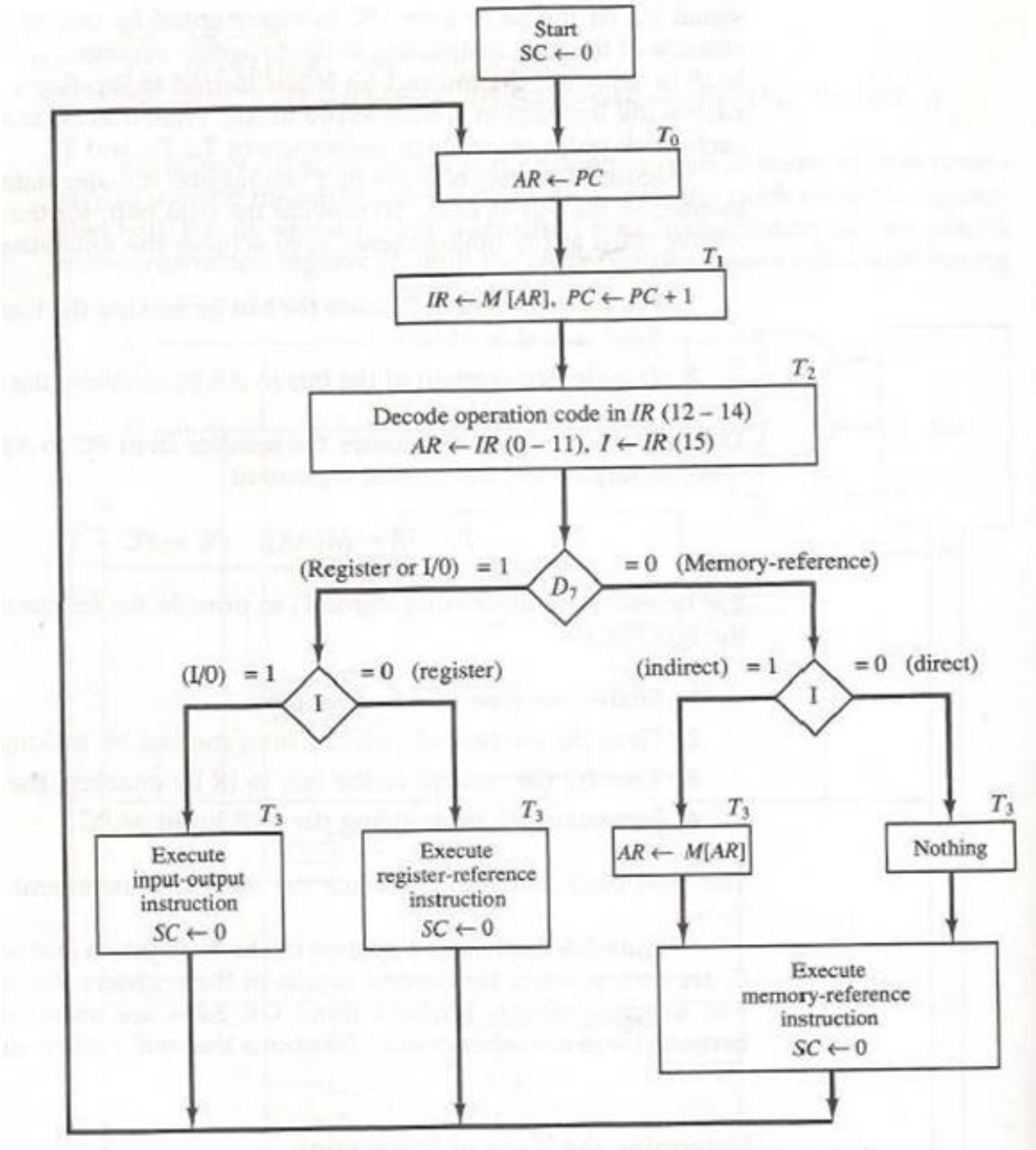
MultipleinputORgatesareincludedinthediagrambecausethereareothercontrolfunctionsthat will initiate similaroperations.

Determine the Type of Instruction:

- ❖ The timing signal that is active after the decoding isT_3.
- ❖ During time T_3, the control unit determine the type of instruction that was read from thememory.
- ❖ The flowchart of fig. shows the initial configurations for the instruction cycle and also how the control determines the instruction cycle type after thedecoding.
- ❖ Decoder output D_7 is equal to 1 if the operation code is equal to binary111.
- ❖ If D_7=1, the instruction must be a register-reference or input-outputtype.
- ❖ If $D7$ = 0, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.
- ❖ Control then inspects the value of the first bit of the instruction, which is now available in flip-flopI.
- ❖ If $D7$ = 0 and I = 1, indicates a memory-reference instruction with an indirect address. So it is then necessary to read the effective address frommemory.
- ❖ If $D7$ = 0 and I = 0, indicates a memory-reference instruction with a directaddress.
- ❖ If $D7$ = 1 and I = 0, indicates a register-referenceinstruction.
- ❖ If $D7$ = 01and I = 1, indicates an input-outputinstruction.
- ❖ The three instruction types are subdivided into four separatepaths.
- ❖ The selected operation is activated with the clock transition associated with timing signalT_3.

This can be symbolized as follows:

$D_7'IT_3$: $AR \leftarrow M[AR]$
$D_7'I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instruction
D_7IT_3: Execute an input–output instruction



Flowchart for instruction cycle (initial configuration).

$D_7'IT_3$: $AR \leftarrow M[AR]$
$D_7'I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instruction
D_7IT_3: Execute an input–output instruction

Register-Reference Instructions:

Register-reference instructions are recognized by the control when D7 = 1 andI=0.These instructions use bits 0 through 11 of the instruction code to specify one of 12instructions.These 12 bits are available in IR(0-11).The control functions and microoperations for the register-reference instructions are listed in Table.

These instructions are executed with the clock transition associated with timing variableT_3.Control function needs the Boolean relation $D_7I'T_3$, which we designate for convenience by the symbol r.By assigning the symbol $B_i$to bit i of *IR*, all control functions can be simply denoted byrB_i.

Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

For example, the instruction CLA has the hexadecimal code 7800, which gives the binary equivalent 0111 1000 0000 0000. The first bit is a zero and is equivalent toI'.The next three bits constitute the operation code and are recognized from decoder outputD_7.Bit 11 in IR is 1 and is recognized from B_{11}. The control function that initiates the microoperation for this instruction is $D_7I'T_3\ B_{11} = rB_{11}$.

The execution of a register-reference instruction is completed at timeT_3.The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signalT_0.

The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or Eregisters.

Thenextfourinstructionscauseaskipofthenextinstructioninsequencewhen a stated condition is satisfied. The skipping of the instruction is achieved by incrementing *PC* onceagain.The condition control statements must be recognized as part of the controlconditions.The *AC* is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of *AC* is zero *(AC* = 0) if all the flip-flops of the register arezero.The HLT instruction clears a start-stop flip-flop *S* and stops the sequence counter fromcounting.

6. Memory-Reference Instructions:

Below table lists the seven memory-referenceinstructions.The decoded output $D_i$for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs to each instruction is included in thetable.The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I= 0, or during timing signal T_3 when I =1.The execution of the memory-reference instructions starts with timing signalT_4.

The symbolic description of each instruction is specified in the table in terms of register transfer notation.

Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC:

- This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address.
- The result of the operation is transferred toAC.
- The microoperations that execute this instructionare:

$$D_0T_4: \quad DR \leftarrow M[AR]$$
$$D_0T_5: \quad AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0$$

ADD to AC:

- This instruction adds the content of the memory word specified by the effective address to the value of*AC*.
- The sum is transferred into AC and the output carry $C_{out}$is transferred to the E (extended accumulator)flip-flop.
- The microoperations needed to execute this instructionare

$$D_1T_4: \quad DR \leftarrow M[AR]$$
$$D_1T_5: \quad AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \quad SC \leftarrow 0$$

LDA: Load to AC

- This instruction transfers the memory word specified by the effective address toAC.
- The microoperations needed to execute this instructionare

$$D_2T_4: \quad DR \leftarrow M[AR]$$
$$D_2T_5: \quad AC \leftarrow DR, \quad SC \leftarrow 0$$

STA: Store AC

- This instruction stores the content of *AC* into the memory word specified by the effectiveaddress.
- Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with onemicrooperation.

$$D_3T_4: \quad M[AR] \leftarrow AC, \quad SC \leftarrow 0$$

BUN: Branch Unconditionally

- This instruction transfers the program to the instruction specified by the effectiveaddress.
- The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps)unconditionally.
- The instruction is executed with onemicrooperation:

$$D_4T_4: \quad PC \leftarrow AR, \quad SC \leftarrow 0$$

BSA: Branch and Save Return Address

- Thisinstructionisusefulforbranchingtoaportionoftheprogramcalledasubroutineorprocedure.
- Whenexecuted,theBSAinstructionstorestheaddressofthenextinstructioninsequence(whichis available in PC) into a memory location specified by the effectiveaddress.
- The effective address plus one is then transferred to *PC* to serve as the address of the first

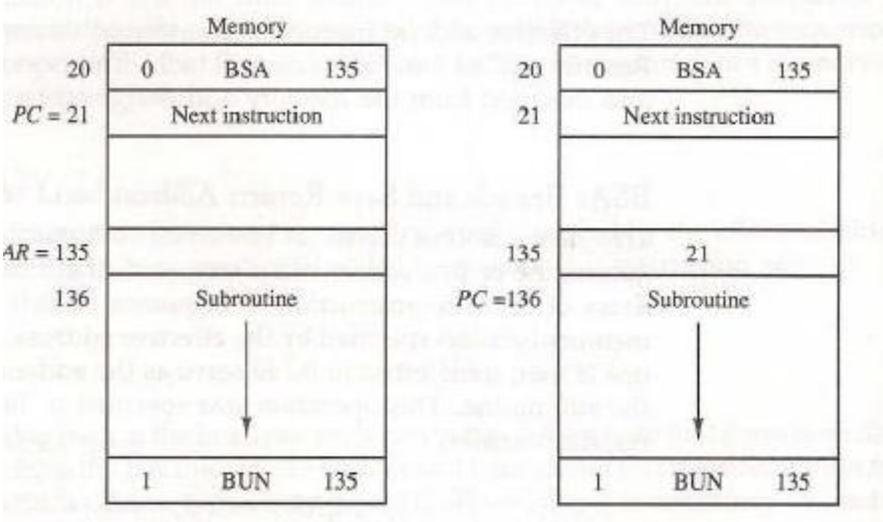
instruction in the subroutine.

❖ This operation was specified with the following registertransfer:

$$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$$

❖ A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig.

Example of BSA instruction execution.



(a) Memory, PC, and AR at time T_4 (b) Memory and PC after execution

The BSA instruction is assumed to be in memory at address20.The I bit is 0 and the address part of the instruction has the binary equivalent of135.After the fetch and decode phases, *PC* contains 21, which is the address of the next instruction in the program (referred to as the return *address*). AR holds the effective address135.This is shown in part (a) of thefigure.

The BSA instruction performs the following numericaloperation:

$$M[135] \leftarrow 21, \quad PC \leftarrow 135 + 1 = 136$$

The result of this operation is shown in part (b) of thefigure.The return address21 is stored in memory location 135 and control continues with the subroutine program starting from address136.Thereturntotheoriginalprogram(ataddress21)isaccomplishedbymeansofanindirectBUN instruction placed at the end of thesubroutine.

When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address21.

When the BUN instruction is executed, the effective address 21 is transferred toPC.The next instruction cycle finds *PC with* the value 21, so control continues to execute the instruction at the returnaddress.

The BSA instruction must be executed with a sequence of twomicrooperations:

$$D_5T_4: \quad M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1$$
$$D_5T_5: \quad PC \leftarrow AR, \quad SC \leftarrow 0$$

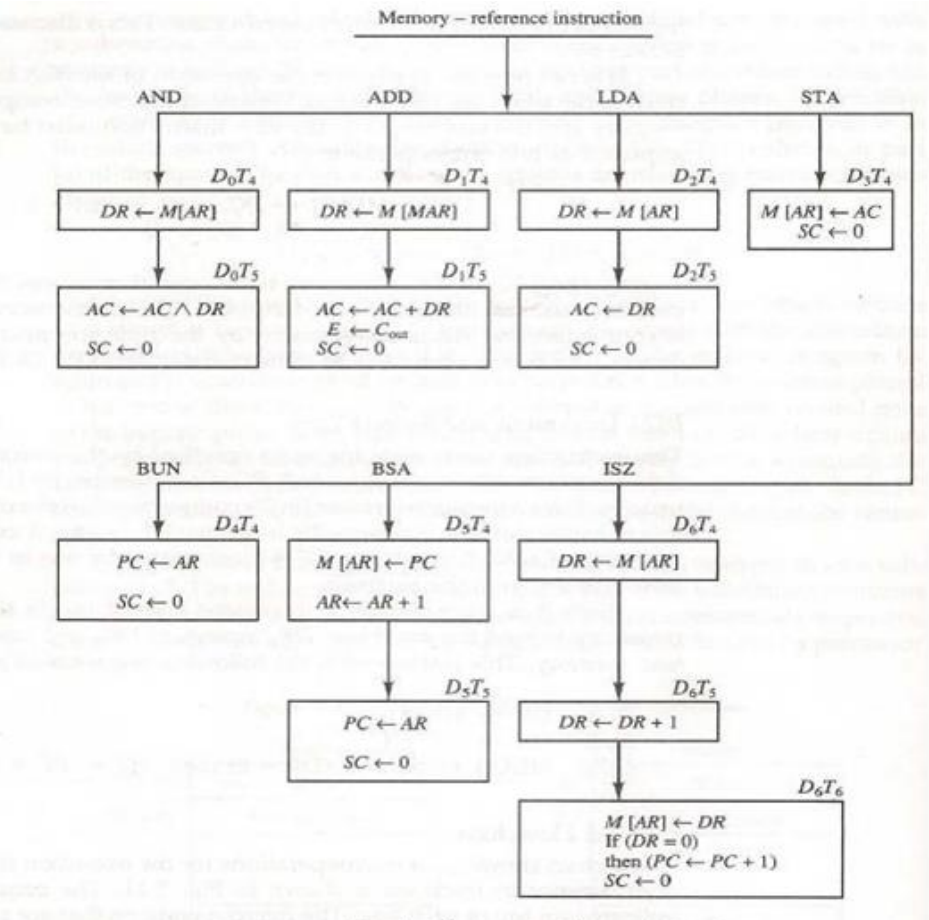
ISZ: Increment and Skip if Zero

- Thisinstructionincrementthewordspecifiedbytheeffectiveaddress,andiftheincrementedvalue is equal to 0, PC is incremented by 1 to skip the next instruction in theprogram.
- Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back intomemory.
- This is done with the following sequence ofmicrooperations:

$$D_6T_4: \quad DR \leftarrow M[AR]$$
$$D_6T_5: \quad DR \leftarrow DR + 1$$
$$D_6T_6: \quad M[AR] \leftarrow DR, \quad \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), \quad SC \leftarrow 0$$

Control Flowchart:

A flowchart showing all microoperations for the execution of the seven memory-reference instructions is shown in Fig.



Flowchart for memory-reference instructions.

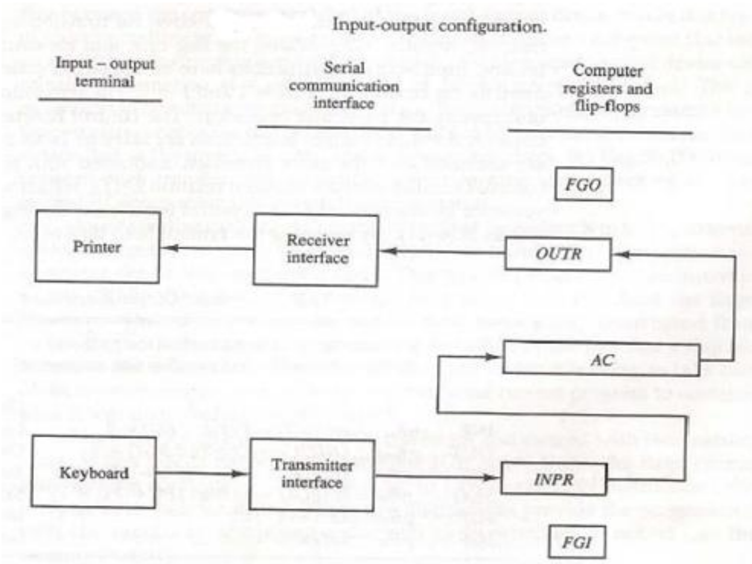
7. Input-Output and Interrupt:

Instructions and data stored in memory must come from some inputdevice.Computational results must be transmitted to the user through some outputdevice.

Todemonstratethemostbasicrequirementsforinputandoutputcommunication,wewilluseasan illustration a terminal unit with a keyboard andprinter.

Input-Output Configuration:

The terminal sends and receives serialinformation.Each quantity of information has eight bits of an alphanumericcode.The serial information from the keyboard is shifted into the input register*INPR*.The serial information for the printer is stored in the output registerOUTR.These two registers communicate with a communication interface serially and with the AC inparallel.The input—output configuration is shown in Fig.



The input register INPR consists of eight bits and holds alphanumeric inputinformation.The 1-bit input flag *FGI* is a controlflip-flop.The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by thecomputer.The output register OUTR works similarly but the direction of information flow isreversed.

Initially, the output flag *FGO* is set to1.The computer checks the flag bit; if it is 1, the information from *AC* is transferred in parallel to OUTR and FGO is cleared to0.The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to1.

Input-Output Instructions:

Input and output instructions are needed for transferring information to and from *AC* register, for checking the flag bits, and for controlling the interruptfacility.Input-outputinstructionshaveanoperationcode1111andarerecognizedbythecontrolwhenD7= 1 and I = 1.The remaining bits of the instruction specify the particularoperation.

The control functions and microoperations for the input-output instructions are listed in Table.

Input-Output Instructions

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR, \quad FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7), \quad FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

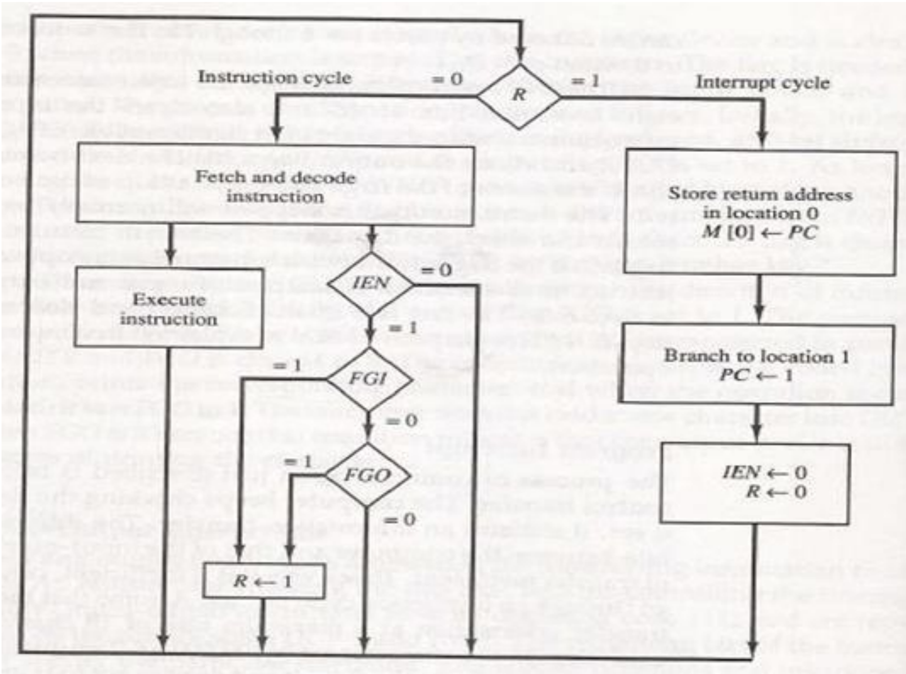
These instructions are executed with the clock transition associated with timing signalT_3.Each control function needs a Boolean relation D_7IT_3, which we designate for convenience by the symbol p.The control function is distinguished by one of the bits in IR(6-11).

By assigning the symbol B_i to bit *i* of IR, all control functions can be denoted by pB_i for i = 6 though 11.The sequence counter *SC* is cleared to 0 when p = D_7IT_3 =1.The last two instructions set and clear an interrupt enable flip-flopIEN.

Program Interrupt:

- Thecomputerkeepscheckingtheflagbit,andwhen itfindsitset,itinitiatesaninformationtransfer.
- Thedifferenceofinformationflowratebetweenthecomputerandthatoftheinput—outputdevice makes this type of transferinefficient.
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for thetransfer.
- In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.
- While the computer is running a program, it does not check theflags.
- When a flag is set, the computer is momentarily interrupted from the currentprogram.
- The computer deviates momentarily from what it is doing to perform of the input or outputtransfer.
- It then returns to the current program to continue what it was doing before theinterrupt.
- The interrupt enable flip-flop IEN can be set and cleared with twoinstructions.
  - ✓ When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt thecomputer.
  - ✓ When *IEN* is set to (with the ION instruction), the computer can beinterrupted.
- Thewaythattheinterruptishandledbythecomputercanbeexplainedbymeansoftheflowchartof Fig.
- An interrupt flip-flop R is included in the computer. When *R* = 0, the computer goes through an instructioncycle.

- During the execute phase of the instruction cycle IEN is checked by thecontrol.
- If it is 0, it indicates that the programmer does not want to use the interrupt,so control continues with the next instructioncycle.
- If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, controlcontinueswith the next instruction cycle.
- If either flag is set to 1 while *1EN* = 1, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.



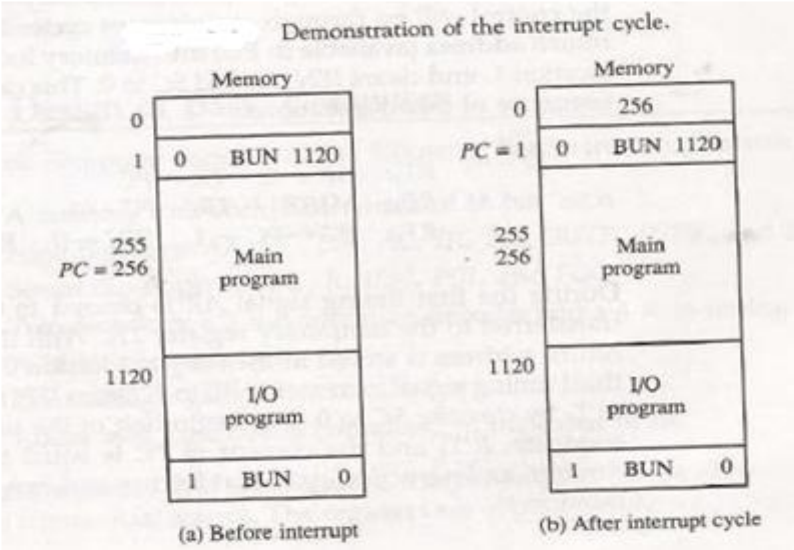
Flowchart for interrupt cycle.

Interrupt cycle:

The interrupt cycle is a hardware implementation of a branch and save return addressoperation.The return address available in PC is stored in a specificlocation.

This location may be a processor register, a memory stack, or a specific memorylocation.

An example that shows what happens during the interrupt cycle is shown in Fig.



When an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255.At this time, the returns address 256 is in*PC.*Theprogrammerhaspreviouslyplacedaninput—outputserviceprograminmemorystartingfrom address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig.(a).When control reaches timing signal $T_0$and finds that $R = 1$, it proceeds with the interruptcycle.The content of *PC* (256) is stored in memory location 0, *PC* is set to 1, and R is cleared to0.Thebranchinstructionataddress1causestheprogramtotransfertotheinput—outputservice program at address1120.This program checks the flags, determines which flag is set, and then transfers the required input or outputinformation.Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it wasinterrupted.This is shown in Fig.(b).

MICRO PROGRAMMED CONTROL

Hardwired Control Unit:

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.

Micro programmed control unit:

A control unit whose binary control variables are stored in memory is called a micro programmed control unit.

Dynamic microprogramming:

A more advanced development known as dynamic microprogramming permits a microprogram to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic microprogramming employ a writable control memory. This type of memory can be used forwriting.

1. Control Memory:

Control Memory is the storage in the microprogrammed control unit to store the microprogram.

Writeable Control Memory:

Control Storage whose contents can be modified, allow the change in microprogram and Instruction set can be changed or modified is referred as Writeable Control Memory**.**

Control Word:

The control variables at any given time can be represented by a control word string of 1 's and 0's called a control word.

Microoperations:

In computer central processing units, micro-operations (also known as a micro-ops or μops) are detailed low-level instructions used in some designs to implement complex machine instructions (sometimes termed macro-instructions in thiscontext).

Micro instruction:

A symbolic microprogram can be translated into its binary equivalent by means of an assembler.Each line of the assembly language microprogram defines a symbolicmicroinstruction.Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, andAD.

Micro program:

A sequence of microinstructions constitutes amicroprogram. Since alterations of the microprogram are not needed once the control unit is in operation, the control memory can be a read-only memory(ROM).ROM words are made permanent during the hardware production of theunit.

The use of a micro program involves placing all control variables in words of ROM for use by the control unit through successive readoperations.

The content of the word in ROM at a given address specifies amicroinstruction.

Microcode:

Microinstructions can be saved by employing subroutines that use common sections of microcode.For example, the sequence of micro operations needed to generate the effective address of the operand for an instruction is common to all memory referenceinstructions.This sequence could be a subroutine that is called from within many other routines toexecute the effective addresscomputation.

Organization of micro programmed control unit

The general configuration of a micro-programmed control unit is demonstrated in the block diagram of Figure.

The control memory is assumed to be a ROM, within which all control information is permanentlystored.

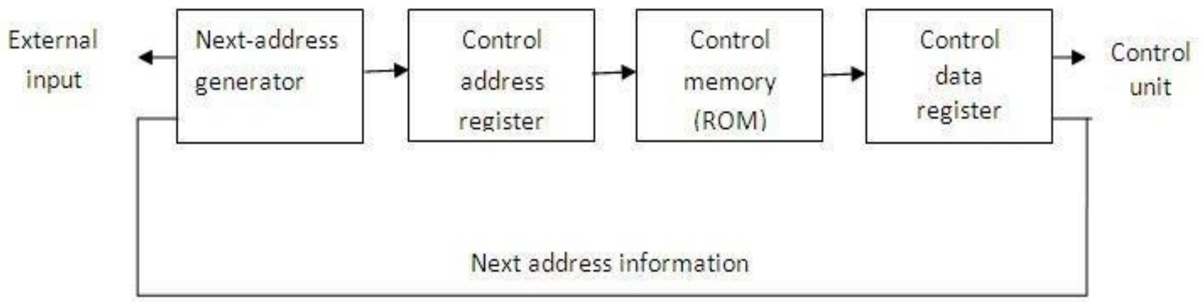


Figure: Micro-programmed control organization

The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read frommemory.

The microinstruction contains a control word that specifies one or more microoperationsfor the data processor. Once these operations are executed, the control must determine the nextaddress.

The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the controlmemory.

While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the nextmicroinstruction.

Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the controlmemory.The next address generator is sometimes called a **micro-program sequencer**, as itdetermines the address sequence that is read from controlmemory.

Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.The control data register holds the present microinstruction while the next address iscomputed and read frommemory.

The data register is sometimes called a **pipelineregister**.

It allows the execution of the microoperations specified by the control word simultaneously with the generation of the nextmicroinstruction.

This configuration requires a two-phase clock, with one clock applied to the address register and the other to the dataregister.

The main advantage of the micro programmed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.

If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for controlmemory.

2. Address Sequencing:

Microinstructions are stored in control memory in groups, with each group specifyingaroutine.

To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computerinstruction.

Step-1:

- An initial address is loaded into the control address register when power is turned on in thecomputer.
- This address is usually the address of the first microinstruction that activates the instruction fetchroutine.
- The fetch routine may be sequenced by incrementing the control address register through the rest of itsmicroinstructions.
- At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2:

- The control memory next must go through the routine that determines the effective address of theoperand.
- A machine instruction may have bits that specify various addressing modes, such as indirect address and indexregisters.
- The effective address computation routine in control memory can be reached through a

branch microinstruction, which is conditioned on the status of the mode bits of the instruction.

- When the effective address computation routine is completed, the address of the operand is available in the memory addressregister.

Step-3:

- The next step is to generate the microoperations that execute the instruction fetched from memory.
- The microoperation steps to be generated in processor registers depend on the operation code part of theinstruction.
- Each instruction has its own micro-program routine stored in a given location of control memory.
- The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a **mapping**process.
- A mapping procedure is a rule that transforms the instruction code into a control memoryaddress.

Step-4:

- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control addressregister.
- Micro-programs that employ subroutines will require an external register for storing the returnaddress.
- Return addresses cannot be stored in ROM because the unit has no writingcapability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first address of the fetchroutine.

In summary, the address sequencing capabilities required in a control memory are:

- Incrementing of the control addressregister.
- Unconditional branch or conditional branch, depending on status bitconditions.
- A mapping process from the bits of the instruction to an address for controlmemory.
- A facility for subroutine call andreturn.

Selection of address for control memory

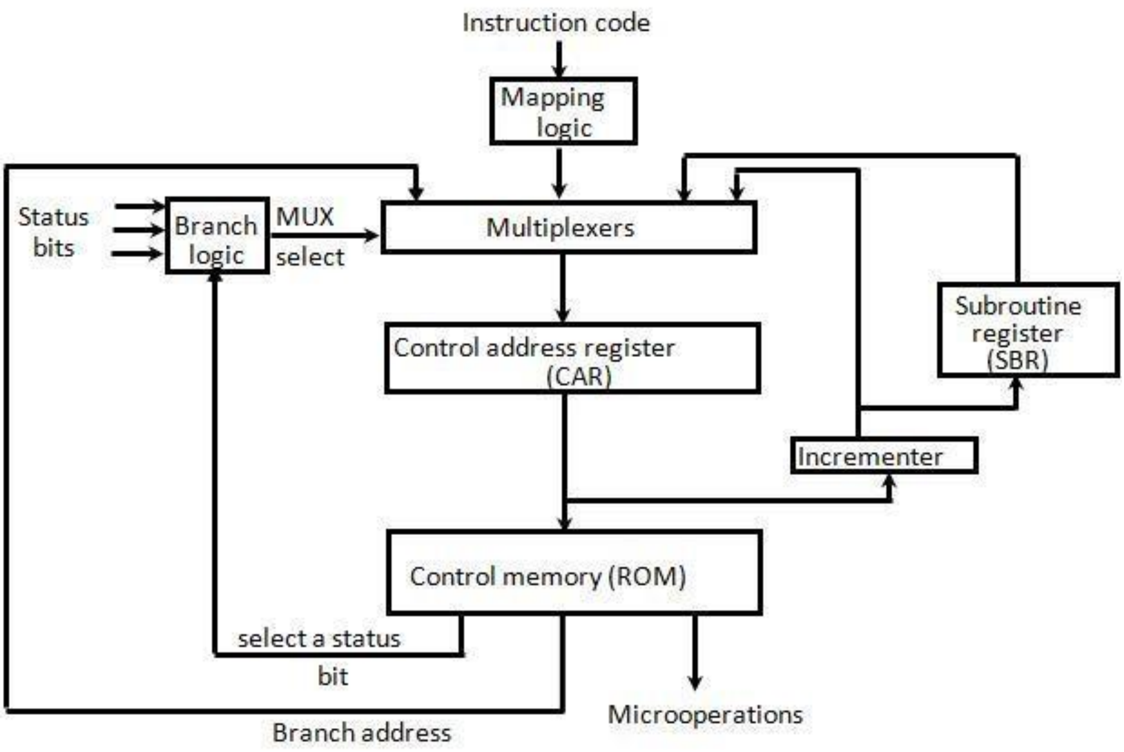


Figure: Selection of address for controlmemory

Above figure shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstructionaddress.

The microinstruction in control memory contains a set of bits to initiate microoperationsin computer registers and other bits to specify the method by which the next address is obtained.

The diagram shows four different paths from which the control address register (CAR) receives theaddress.The incrementer increments the content of the control address register by one, to select the next microinstruction insequence.

Branching is achieved by specifying the branch address in one of the fields of the microinstruction.Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine itscondition.

An external address is transferred into control memory via a mapping logiccircuit.

The return address for a subroutine is stored in a special register whose value is then used when the micro-program wishes to return from thesubroutine.

The branch logic of figure provides decision-making capabilities in the controlunit.

The status conditions are special bits in the system that provide parameter informationsuch as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output statusconditions.

The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branchlogic.

A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control addressregister.

A 0 output in the multiplexer causes the address register to beincremented.

Mapping of an Instruction

A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction islocated.

The status bits for this type of branch are the bits in the operation code part of the instruction.

For example, a computer with a simple instruction format as shown in figure has an operation code of four bits which can specify up to 16 distinct instructions.

Assume further that the control memory has 128 words, requiring an address of seven bits.

One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure.

This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control addressregister.

This provides for each computer instruction a microprogram routine with a capacity of fourmicroinstructions.

If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for otherroutines.

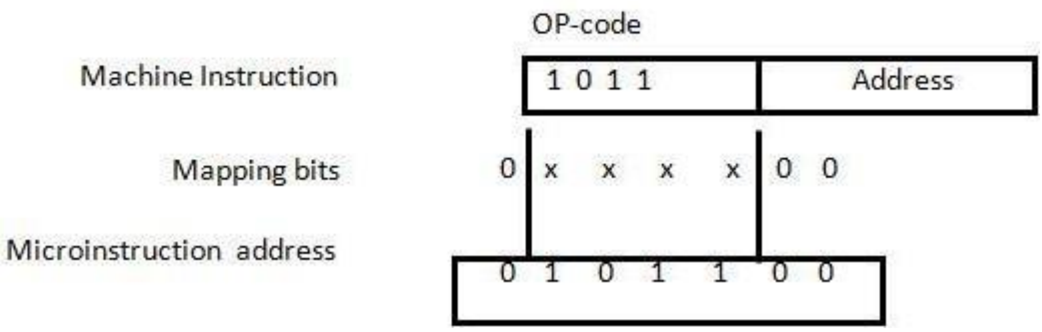


Figure: Mapping from instruction code to microinstruction address

One can extend this concept to a more general mapping rule by using a ROM to specify the mappingfunction.

The contents of the mapping ROM give the bits for the control addressregister.

In this way the microprogram routine that executes the instruction can be placed in any desired location in controlmemory.

The mapping concept provides flexibility for adding instructions for control memory as the needarises.

Computer Hardware Configuration

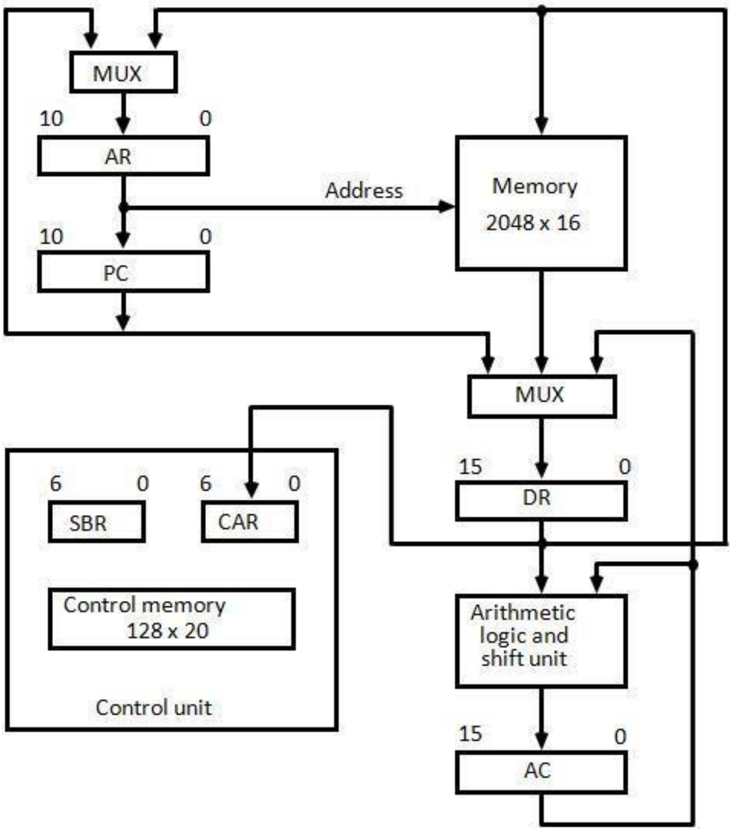


Figure: Computer hardware configuration

The block diagram of the computer is shown in Figure. It consists of

Two memoryunits:

Main memory -> for storing instructions and data, and Control memory -> for storing the microprogram.

SixRegisters:

Processor unit register: AC(accumulator),PC(Program Counter), AR(Address Register), DR(Data Register)

Control unit register: CAR (Control Address Register), SBR(Subroutine Register)

Multiplexers:

The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

ALU:

The arithmetic, logic, and shift unit performs microoperations with data from AC and DR and places the result in AC.

DR can receive information from AC, PC, ormemory.

AR can receive information from PC orDR.

PC can receive information only fromAR.

Input data written to memory come from DR, and data read from memory can go only toDR.

Microinstruction Format

The microinstruction format for the control memory is shown in figure 4.5. The 20 bits of the microinstruction are divided into four functional parts as follows:

The three fields F1, F2, and F3 specify microoperations for thecomputer.

The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. This gives a total of 21 microoperations.

The CD field selects status bitconditions.

The BR field specifies the type of branch to beused.

The AD field contains a branch address. The address field is seven bits wide, sincethe control memory has 128 = 2^7words.

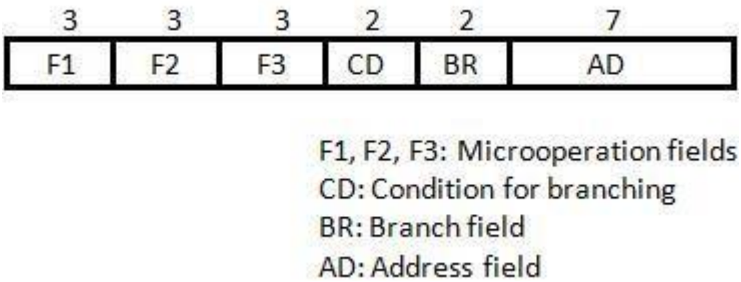


Figure: Microinstruction Format

As an example, a microinstruction can specify two simultaneous microoperations from F2 and F3 and none fromF1.

DR M[AR] with F2 = 100 PC PC + 1 with F3 = 101

The nine bits of the microoperation fields will then be 000 100101.

The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table.

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

Table: Condition Field

The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction shown in Table4.2.

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1 CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1 CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Table: Branch Field

Symbolic Microinstruction.

Each line of the assembly language microprogram defines a symbolicmicroinstruction.

Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD. The fields specify the following Table4.3.

1.	Label	The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
2.	Microoperations	It consists of one, two, or three symbols, separated by commas, from those defined in Table 5.3. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations. This will be translated by the assembler to nine zeros.

3.	CD	The CD field has one of the letters U, I, S, or Z.
4.	BR	The BR field contains one of the four symbols defined in Table 5.2.
5.	AD	The AD field specifies a value for the address field of the microinstruction in one of three possible ways: i. With a symbolic address, this must also appear asa label. ii. With the symbol NEXT to designate the next address insequence. iii. When the BR field contains a RET or MAP symbol, the AD field is left empty and is converted toseven zeros by the assembler.

Table: Symbolic Microinstruction

Micro programmed sequencer for a control memory:

Microprogram sequencer:

The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.

The address selection part is called a microprogramsequencer.

A microprogram sequencer can be constructed with digital functions to suit a particular application.

To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range ofapplications.

The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read andexecuted.

Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutinecalls.

Some sequencers provide an output register which can function as the address register for the controlmemory.The block diagram of the microprogram sequencer is shown in figure.

There are two multiplexers in thecircuit.

The first multiplexer selects an address from one of four sources and routes it into acontrol address registerCAR.

The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.

The output from CAR provides the address for the controlmemory.

The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine registersSBR.

The other three inputs to multiplexer 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction.

Although the figure 4.6 shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep. In this way, a number of subroutines can be active at the sametime.

The CD (condition) field of the microinstruction selects one of the status bits in the secondmultiplexer.

If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise, it is equal to0.

The T value together with the two bits from the BR (branch) field goes to an input logic circuit.

The input logic in a particular sequencer will determine the type of operations that are available in theunit.

BR	**Input**			**MUX 1**		**Load SBR**
	I1	**I0**	**T**	**S1**	**S0**	**L**
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	0	1	1
1 0	1	0	X	1	0	0
1 1	1	1	X	1	1	0

Table: Input Logic Truth Table for Microprogram Sequencer

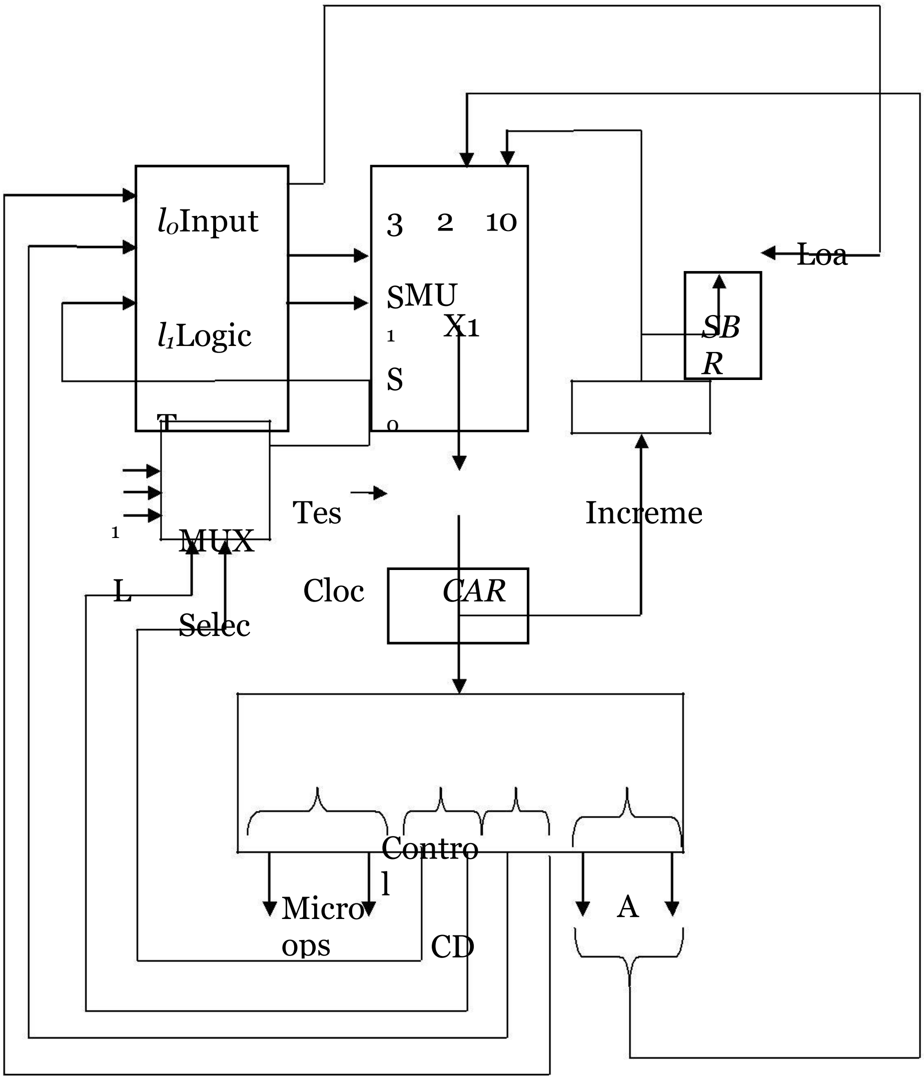


Figure: Microprogram Sequencer for a control memory

Boolean Function:

S0 = I0

S1 = I0I1 + I0'T L = I0'I1T

Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations.

With three inputs, the sequencer can provide up to eight address sequencingoperations.

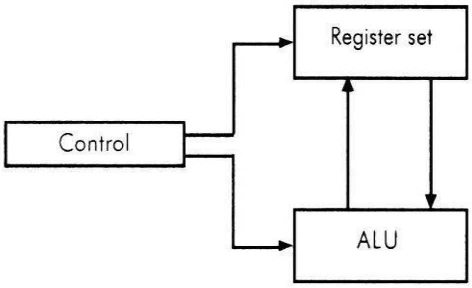
Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range ofoperations.

UNIT – 02 – Part - B

1. General Register Organization:

The Central Processing Unit (CPU) is called the brain of the computer that performsdata-processing operations. Figure 3.1 shows the three major parts of CPU.

.



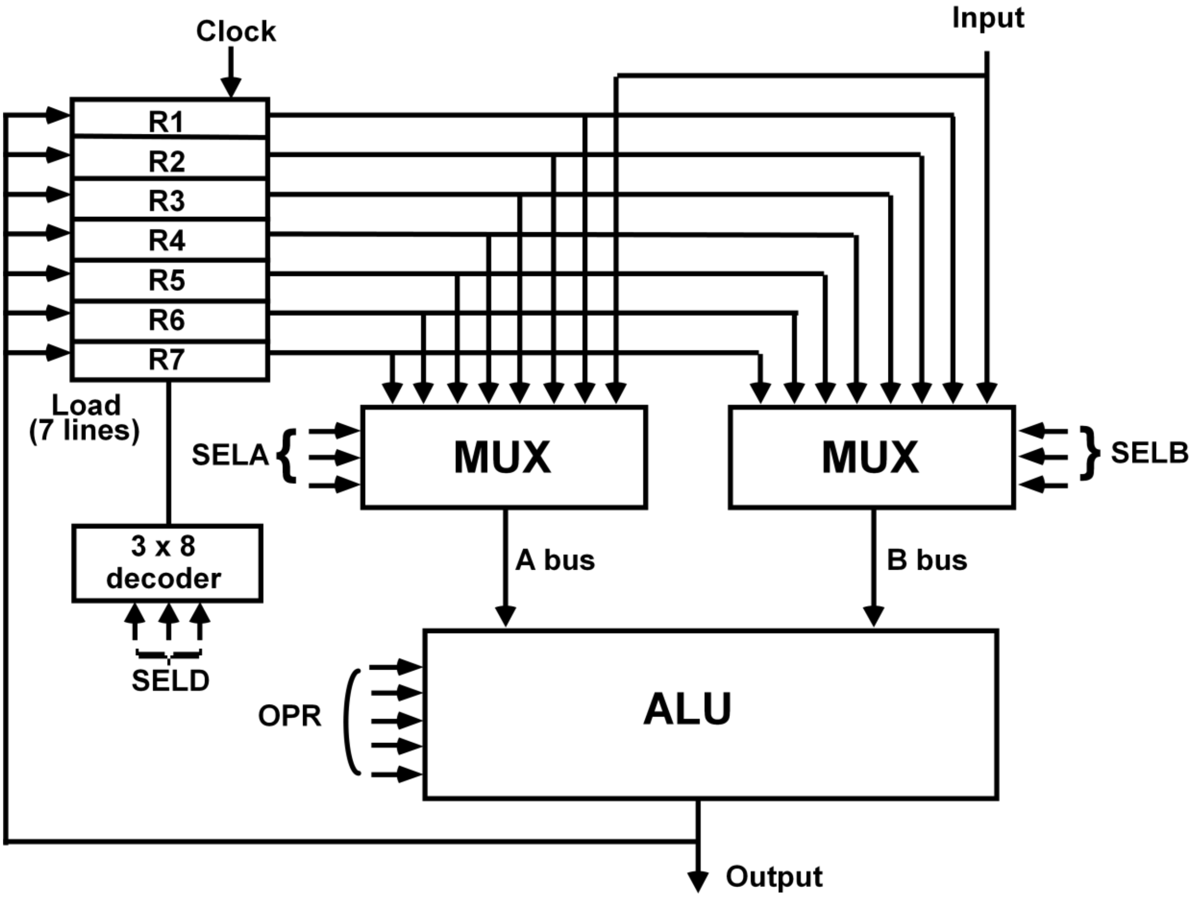
Intermediate data is stored in the register set during the execution of the instructions. The microoperations required for executing the instructions are performed by the arithmetic logic unit whereas the control unit takes care of transfer of information among the registers and guides the ALU. The control unit services the transfer of information among the registers and instructs the ALU about which operation is to be performed. The computer instruction set is meant for providing the specifications for the design of the CPU.

The design of the CPU largely, involveschoosing the hardware for implementing the machine instructions.

The need for memory locations arises for storing pointers, counters, returnaddress, temporary results and partial products. Memory access consumes the most of the time off an operation in a computer. It is more convenient and more efficient to store these intermediate values in processor registers.

A common bus system is employed to contact registers that are included in the CPU in a large number. Communications between registers is not only for direct data transfer but also for performing various micro-operations. A bus organization for such CPU register shown in Figure 3.2, is connected to two multiplexers (MUX) to form two buses A and B. The selected lines in each multiplexers select one register of the input data for the particularbus.

.



OPERATION OF CONTROLUNIT:

The control unit directs the information flow through ALU by:

- ❖ Selecting various Components in thesystem
- ❖ Selecting the Function ofALU

Example: R1 <- R2 + R3

[1] MUX A selector (SELA): BUS A□ R2

[2] MUX B selector (SELB): BUS B □R3

[3] ALU operation selector (OPR): ALU toADD

[4] Decoder destination selector (SELD): R1 □ OutBus

Control Word

3	3	3	5
SELA	SELB	SELD	OPR

Encoding of register selection fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Encoding of ALU operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of ALU Microoperations

Symbolic Designation

Microoperation	**SELA**	**SELB**	**SELD**	**OPR**	**Control Word**
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	-	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	-	R7	TSFA	001 000 111 00000
Output $\leftarrow$ R2	R2	-	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	-	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl R4	R4	-	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

Stack organization:

A stack is a storage device that stores information in such a manner that the item stored last is the first itemretrieved.

The stack in digital computers is essentially a memory unit with an address register that can count only. The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in thestack.

The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted ordeleted.

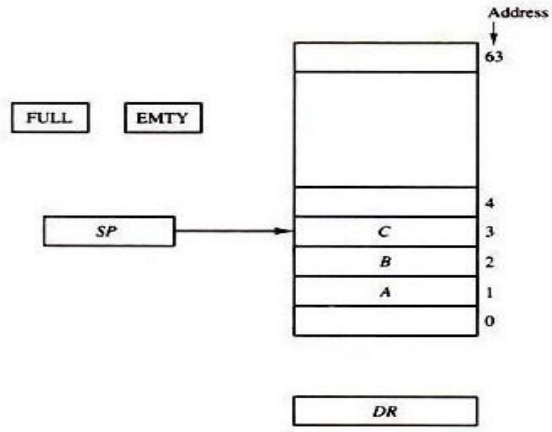


Figure: Block diagram of a 64-word stack

Register stack:

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure shows the organization of a 64- word registerstack.

The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now3.

To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2.

To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in thestack.

In a 64-word stack, the stack pointer contains 6 bits because 2^6 =64.

Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 are incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significantbits.

Similarly, when 000000 is decremented by 1, the result is 111111. The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty ofitems.

DR is the data register that holds the binary data to be written into or read out of thestack.

PUSH:

If the stack is not full (FULL =0), a new item is inserted with a push operation. The push operation consists of the following sequences ofmicrooperations:

SP ← SP+1 Increment stack pointer

M [SP]←DR WRITE ITEM ON TOP OF THESTACK

IF (SP = 0) then (FULL ← 1)Check is stack is full EMTY← 0 Mark the stack notempty

The stack pointer is incremented so that it points to the address of next-higher word. A memory write operation inserts the word from DR into the top of thestack.

SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available inSP.

The first item stored in the stack is at address 1. The last item is stored at address 0. If SPreaches 0, the stack is full of items, so FULL is set to 1. This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location0.

Once an item is stored in location 0, there are no more empty registers in the stack. If anitem is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to0.

POP:

A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequences ofmicrooperations:

DR ←M[SP] Read item on top of thestack

SP ← SP -1 Decrement stack pointer IF (SP = 0) then (EMTY ← 1)

Check if stack is empty FULL← 0 Mark the stack notfull

The top item is read from the stack into DR. The stack pointer is then decremented. If itsvalue reaches zero, the stack is empty, so EMTY is set to1.

This condition is reached if the item read was in location1.

Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. If a pop operation reads the item from location 0 and then SP is decremented, SP is changes to 111111, which is equivalent to decimal63.

In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY =1.

Memory Stack.

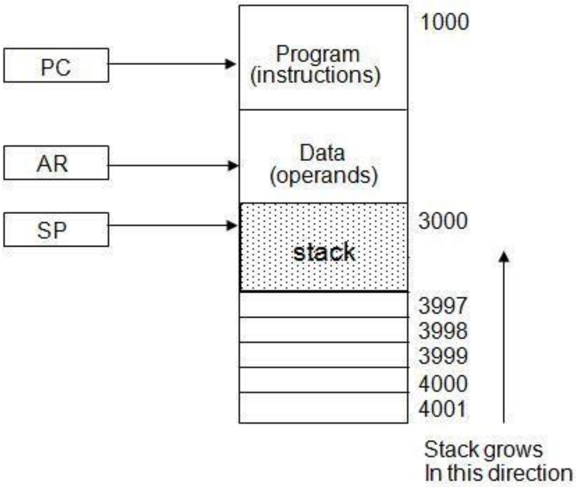


Figure: Computer memory with program, data, and stacksegments

The implementation of a stack in the CPU is done by assigning a portion of memory to astack operation and using a processor register as a stackpointer.

Figure shows a portion of computer memory partitioned into three segments: program, data, andstack.

The program counter PC points at the address of the next instruction in the program which is used during the fetch phase to read aninstruction.

The address registers AR points at an array of data which is used during the execute phase to read anoperand.

The stack pointer SP points at the top of the stack which is used to push or pop items into or from thestack.

The three registers are connected to a common address bus, and either one can provide anaddress formemory.

As shown in Figure, the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is3000.

We assume that the items in the stack communicate with a data registerDR.

PUSH

A new item is inserted with the push operation as follows:

SP ← SP - 1 M[SP] ← DR

The stack pointer is decremented so that it points at the address of the nextword.

A memory write operation inserts the word from DR into the top of thestack.

POP

A new item is deleted with a pop operation asfollows:

$$DR \leftarrow M[SP] SP \leftarrow SP + 1$$

The top item is read from the stack intoDR.

The stack pointer is then incremented to point at the next item in thestack.

The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updatingSP.

Which of the two microoperations is done first and whether SP is updated by incrementing or decrementing depends on the organization of thestack.

In figure the stack grows by decreasing the memory address. The stack may be constructed to grow by increasing the memoryalso.

The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

2. Instruction formats:

Insruction fields:

OP-code field - specifies the operation to be performed

Address field - designates memory address(s) or a processor register(s)

Modefield - specifies the way the operand or the effective address isdetermined.

The number of address fields in the instruction format depends on the internal organization of CPU

-The three most common CPU organizations:

Single accumulator organization:

```
ADD    X            /* AC ← AC + M[X]  */
```

General register organization:

```
ADD    R1, R2, R3   /* R1 ← R2 + R3  */
ADD    R1, R2       /* R1 ← R1 + R2  */
MOV    R1, R2       /* R1 ← R2  */
ADD    R1, X        /* R1 ← R1 + M[X]  */
```

Stack organization:

```
PUSH   X            /* TOS ← M[X]  */
ADD
```

Three-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
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       */
```

- Results in short programs
- Instruction becomes long (many bits)

Two-Address Instructions:

Program to evaluate X = (A + B) * (C + D) :

```
MOV    R1, A        /* R1 ← M[A]          */
ADD    R1, B        /* R1 ← R1 + M[B]     */
MOV    R2, C        /* R2 ← M[C]          */
ADD    R2, D        /* R2 ← R2 + M[D]     */
MUL    R1, R2       /* R1 ← R1 * R2       */
MOV    X, R1        /* M[X] ← R1          */
```

ONE, and ZERO-ADDRESS INSTRUCTIONS

One-Address Instructions:

- Use an implied AC register for all data manipulation
- Program to evaluate X = (A + B) * (C + D) :

```
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          */
```

Zero-Address Instructions:

- Can be found in a stack-organized computer
- Program to evaluate X = (A + B) * (C + D) :

```
PUSH    A    /* TOS ← A                  */
PUSH    B    /* TOS ← B                  */
ADD          /* TOS ← (A + B)            */
PUSH    C    /* TOS ← C                  */
PUSH    D    /* TOS ← D                  */
ADD          /* TOS ← (C + D)            */
MUL          /* TOS ← (C + D) * (A + B)  */
POP     X    /* M[X] ← TOS               */
```

cpe 252: Computer Organization 12

3. Addressing Modes :

Specifies a rule for interpreting or modifying the address field of the instruction (before the operandis actuallyreferenced)

Variety of addressingmodesto give programming flexibility to theuserto use the bits in the address field of the instructionefficiently.

Implied Mode

Address of the operands are specified implicitly in the definition of the instruction

- No need to specify address in the instruction
- EA = AC, or EA = Stack[SP], **EA: Effective Address.**

Immediate Mode

Instead of specifying the address of the operand, operand itself is specified

- No need to specify address in the instruction
  - However, operand itself needs to be specified
  - Sometimes, require more bits than the address
  - Fast to acquire an operand

Register Mode

Address specified in the instruction is the register address

- Designated operand need to be in a register
- Shorter address than the memory address
- Saving address field in the instruction
- Faster to acquire an operand than the memory addressing
- EA = IR(R) (IR(R): Register field of IR)

Types of Addressing Modes :

Register Indirect Mode

Instruction specifies a register which contains the memory address of the operand

Saving instruction bits since register addres is shorter than the memoryaddress

Slower to acquire an operand than both the register addressing or memoryaddressing

EA = [IR(R)] ([x]: Content ofx)

Auto-increment or Auto-decrement features:

Same as the Register Indirect, but:

When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 (after or before the execution of theinstruction)

Direct Address Mode

Instruction specifies the memory address which can be used directly to the physical memory

Faster than the other memory addressingmodes

Too many bits are needed to specify the address for a large physical memoryspace

EA = IR(address), (IR(address): address field ofIR)

Indirect Addressing Mode

The address field of an instruction specifies the address of a memory location that contains the address of the operand

When the abbreviated address is used, large physicalmemorycan be addressed with a relatively small number ofbits

Slow to acquire an operand because of an additional memoryaccess

EA =M[IR(address)]

Relative Addressing Modes

The Address fields of an instruction specifies the part of the address(abbreviated address) which can be used along with a designated register to calculate the address of theoperand

PC Relative Addressing Mode(R = PC)

EA = PC + IR(address)

Address field of the instruction isshort

Large physical memory can be accessed with a small number of addressbits

Indexed Addressing Mode

XR: Index Register:

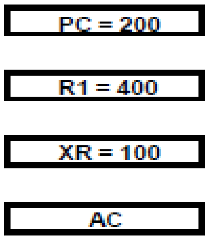
EA = XR + IR(address)

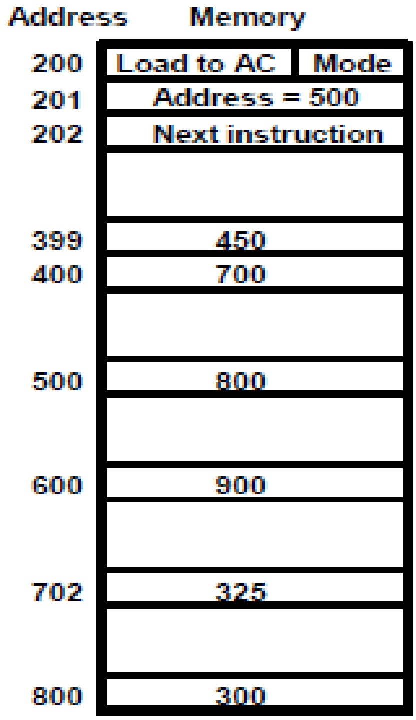
Base Register Addressing Mode

BAR: Base Address Register:

EA = BAR + IR(address)

ADDRESSINGMODES –EXAMPLES:





Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (XR+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Autoincrement	400	/* AC ← (R1)+ */	700
Autodecrement	399	/* AC ← -(R) */	450

cpe 252: Computer Organization

4. Data Transfer Instructions:

Data transfer instructions move data from one place in the computer to another without changing the datacontent.

The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registersthemselves.

The **load** instruction has been used mostly to designate a transfer from memory to a processor register, usually anaccumulator.

The **store** instruction designates a transfer from a processor register intomemory.

The **move** instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memorywords.

The **exchange** instruction swaps information between two registers or a register and a memoryword.

The **input and output** instructions transfer data among processor registers and input or outputterminals.

The **push and pop** instructions transfer data between processor registers and a memory stack.

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

5. Data Manipulation Instructions:

ThreeBasicTypes: **Arithmeticinstructions**

Logical and bit manipulation instructions

Shift instructions

DATA TRANSFER INSTRUCTIONS

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Transfer Instructions with Different Addressing Modes

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1)+	AC ← M[R1], R1 ← R1 + 1
Autodecrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

cpe 252: Computer Organization 19

Arithmetic Instructions:

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate(2's Complement)	NEG

LogicalandBitManipulationInstructions:

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enableinterrupt	EI
Disableinterrupt	DI

cpe252

ShiftInstructions:

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR

Rotate left	**ROL**
Rotate right thru carry	**RORC**
Rotate left thru carry	**ROLC**

6. Program Control Instructions :

It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions be stored for further analysis. Status bits are also called condition-code bits or flagbits.

Figure shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by C, S, Z, and V. The bits are set or cleared as a result of an operation performed in theALU.

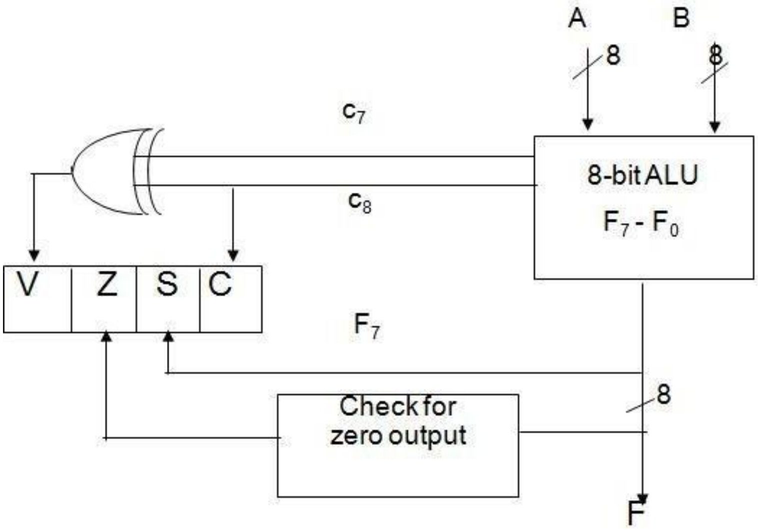


Figure: Status Register Bits

- Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is0.
- Bit S (sign) is set to 1 if the highest-order bit F7 is 1. It is set to 0 if set to 0 if the bit is 0.

- Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. it is cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
- Bit V (overflow) is set to 1 if the exclusives-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement. For the 8-bit ALU, V = 1 if the output is greater than + 127 or less than-128.
- The status bits can be checked after an ALU operation to determine certain relationships that exist between the vales of A and B.
- If bit V is set after the addition of two signed numbers, it indicates an overflowcondition.
- If Z is set after an exclusive-OR operation, it indicates that A =B.
- A single bit in A can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the Z statusbit.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by -)	CMP
Test (by AND)	TST

Program Control Instructions

CMP and TST instructions do not retain their results of operations(- and AND, respectively). They only set or clear certain Flags.

ConditionalBranch Instructions:

MnemonicBranchcondition Testedcondition		
BZ	Branchifzero	Z =1
BNZ	Branch ifnotzero	Z =0
BC	Branchifcarry	C =1
BNC	Branch ifnocarry	C =0
BP	Branchifplus	S =0
BM	Branchifminus	S =1
BV	Branchifoverflow	V =1
BNV	Branchifnooverflow	V =0
Unsigned compare conditions (A - B)		
BHI	Branchifhigher	A >B
BHE	Branchifhigherorequal	A □B
BLO	Branchiflower	A <B
BLOE	Branchiflowerorequal	A □B
BE	Branchifequal	A =B
BNE	Branch ifnotequal	A □B
Signed compare conditions (A - B)		
BGT	Branch ifgreaterthan	A >B
BGE	Branch if greater or equal	A □ B
BLT	Branch iflessthan	A <B
BLE	Branch if lessorequal	A □B
BE	Branchifequal	A =B
BNE	Branch ifnotequal	A □B

Subroutine Call and Return:

Two Most Important Operations are Implied;

Branch to the beginning of the Subroutine

Same as the Branch or Conditional Branch

Save the Return Address to get the address of the location in the Calling Program upon exit from theSubroutine

```
CALL
  SP ←SP -1
  M[SP] ←PC
  PC←EA

RTN
  PC ←M[SP]
  SP ←SP + 1
```

Locations for storing ReturnAddress:

Fixed Location in thesubroutine(Memory)

Fixed Location inmemory

In a processorRegister

In a memorystack

7. Program Interrupt:

The concept of program interrupt is used to handle a variety of problems that arise out of normal programsequence.

Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program isexecuted.

After a program has been interrupted and the service routine been executed, the CPU must return to exactly the same state that it was when the interruptoccurred.

Only if this happens will the interrupted program be able to resume exactly as if nothing hadhappened.

The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determinedfrom:

- ✓ The content of the programcounter
- ✓ The content of all processorregisters
- ✓ The content of certain statusconditions

The interrupt facility allows the running program to proceed until the input or output device sets its ready flag. Whenever a flag is set to 1, the computer completes the execution of the instruction in progress and then acknowledges theinterrupt.

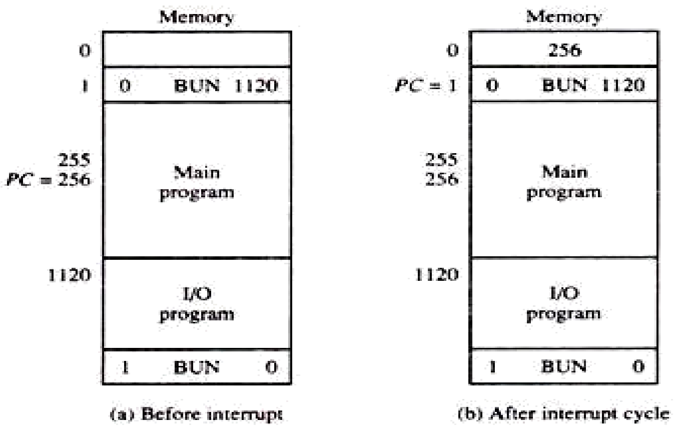
The result of this action is that the retune address is stared in location 0. The instruction in location 1 is then performed; this initiates a service routine for the input or output transfer. The service routine can be stored in location1.

The service routine must have instructions to perform the followingtasks:

- ✓ Save contents of processorregisters.
- ✓ Check which flag isset.
- ✓ Service the device whose flag isset.
- ✓ Restore contents of processorregisters.

✓ Turn the interrupt facilityon.

✓ Return to the runningprogram.



Types of interrupts.:

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

- Externalinterrupts
- Internalinterrupts
- Softwareinterrupts

Externalinterrupts:

External interrupts come from input-output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other externalsource.

Examples that cause external interrupts are I/0 device requesting transfer of data, I/odevice finished transfer of data, elapsed time of an event, or power failure. Timeout interrupt may result from a program that is in an endless loop and thus exceeded its timeallocation.

Power failure interrupt may have as its service routine a program that transfers the complete state of the CPU into a nondestructive memory in the few milliseconds before powerceases.

External interrupts are asynchronous. External interrupts depend on external conditions that are independent of the program being executed at thetime.

Internalinterrupts:

Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also calledtraps.

Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation. These error conditions usually occur as a result of a premature termination of the instruction execution. The service program that processes the internal interrupt determines the corrective measure to betaken.

Internal interrupts are synchronous with the program. . If the program is rerun, the internal interrupts will occur in the same place eachtime.

Softwareinterrupts:

A software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in theprogram.

The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure. A program written by a user must run in the usermode.

When an input or output transfer is required, the supervisor mode is requested by means of asupervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisormode.

The calling program must pass information to the operating system in order to specify the particular taskrequested.

Reverse Polish Notation (RPN) with appropriateexample.

The postfix RPN notation, referred to as Reverse Polish Notation (RPN), places the operator after theoperands.

The following examples demonstrate the threerepresentations:

A +B	Infixnotation
+AB	Prefix or Polishnotation
A B+	Postfix or reverse Polishnotation

The reverse Polish notation is in a form suitable for stackmanipulation.

The expression

A * B + C * D is written in reverse Polish notation as A B * C D * +

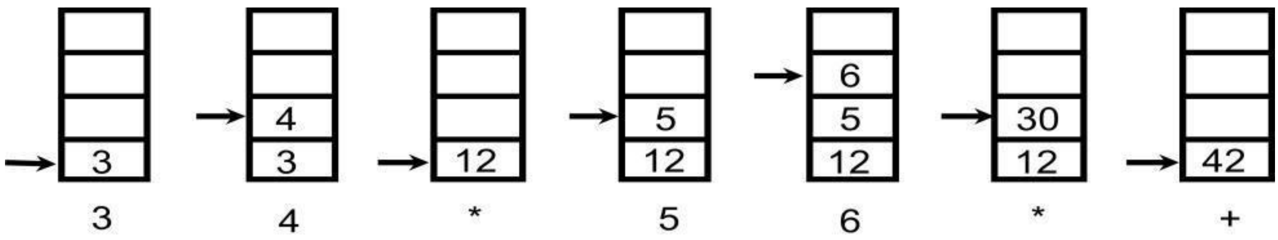
The conversion from infix notation to reverse Polish notation must take into consideration the operational hierarchy adopted for infixnotation.

This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operations before addition and subtractionoperations.

Evaluation of Arithmetic Expressions

Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polishnotation

(3 * 4) + (5 *6) □ 3 4 * 5 6 * +



UNIT – 03 – Part - A

DATA REPRESENTATION

3.1 Computer Data types:

Computer programs or application may use *different types of data* based on the problem or requirement.

Given below is different *types of data that computer uses*:

- **Numeric data** – Integer and Real numbers
- **Non-numeric data** – Character data, address data, logical data

Let's study about each with further sub-categories.

Numeric data:

It can be of the following two types:

- ✓ Integers
- ✓ Real Numbers

Real numbers can be represented as:

1. Fixed point representation
2. Floating point representation

Character data:

A sequence of character is called *character data*.

A character may be alphabetic (A-Z or a-z), numeric (0-9), special character (+, #, *, @, etc.) or combination of all of these. A character is represented by group of bits.

When set of multiple character are combined together they form a meaningful data. A character is represented in standard ASCII format.Another popular format is EBCDIC used in large computer systems.

Example of character data

- ➢ Rajneesh1#
- ➢ 229/3, xyZ
- ➢ Mission Milap – X/10

Logical data

A logical data is used by computer systems to take logical decisions.

Logical data is different from numeric or alphanumeric data in the way that numeric and alphanumeric data may be associated with numbers or characters but logical data is denoted by either of two values true (T) or false(F).

You can see the example of logical data in construction of truth table in logic gates.

A logical data can also be statement consisting of numeric or character data with relational symbols (>, <, =, etc.).

Character set

Character sets can of following types in computers:

- ❖ **Alphabetic characters**- It consists of alphabet characters A-Z or a-z.
- ❖ **Numeric characters**- It consists of digits from 0 to 9.
- ❖ **Special characters**- Special symbols are +, *, /, -, ., <, >, =, @, %, #, etc.

3.2. Number System:

Human beings use *decimal* (base 10) and *duodecimal* (base 12) number systems for counting and measurements (probably because we have 10 fingers and two big toes). Computers use *binary* (base 2) number system, as they are made from binary digital components (known as transistors) operating in two states - on and off. In computing, we also use *hexadecimal* (base 16) or *octal* (base 8) number systems, as a *compact* form for representing binary numbers.

3.2.1 Decimal (Base 10) Number System

Decimal number system has ten symbols: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, and `9`, called *digits*. It uses *positional notation*. That is, the least-significant digit (right-most digit) is of the order of `10^0` (units or ones), the second right-most digit is of the order of `10^1` (tens), the third right-most digit is of the order of `10^2` (hundreds), and so on, where `^` denotes exponent. For example,

735 = 700 + 30 + 5 = 7×10^2 + 3×10^1 + 5×10^0

We shall denote a decimal number with an optional suffix `D` if ambiguity arises.

3.2.2 Binary (Base 2) Number System

Binary number system has two symbols: `0` and `1`, called *bits*. It is also a *positional notation*, for example,

10110B = 10000B + 0000B + 100B + 10B + 0B = 1×2^4 + 0×2^3 + 1×2^2 + 1×2^1 + 0×2^0

We shall denote a binary number with a suffix `B`. Some programming languages denote binary numbers with prefix `0b` or `0B` (e.g., `0b1001000`), or prefix `b` with the bits quoted (e.g., `b'10001111'`).

A binary digit is called a *bit*. Eight bits is called a *byte* (why 8-bit unit? Probably because $8=2^3$).

3.2.3 Hexadecimal (Base 16) Number System

Hexadecimal number system uses 16 symbols: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `A`, `B`, `C`, `D`, `E`, and `F`, called *hex digits*. It is a *positional notation*, for example,

A3EH = A00H + 30H + EH = 10×16^2 + 3×16^1 + 14×16^0

We shall denote a hexadecimal number (in short, hex) with a suffix `H`. Some programming languages denote hex numbers with prefix `0x` or `0X` (e.g., `0x1A3C5F`), or prefix `x` with hex digits quoted (e.g., `x'C3A4D98B'`).

Each hexadecimal digit is also called a *hex digit*. Most programming languages accept lowercase `'a'` to `'f'` as well as uppercase `'A'` to `'F'`.

0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Computers uses binary system in their internal operations, as they are built from binary digital electronic components with 2 states - on and off. However, writing or reading a long sequence of binary bits is cumbersome and error-prone (try to read this binary string: `1011 0011 0100 0011 0001 1101 0001 1000B`, which is the same as hexadecimal `B343 1D18H`). Hexadecimal system is used as a *compact* form or *shorthand* for binary bits. Each hex digit is equivalent to 4 binary bits, i.e., shorthand for 4 bits, as follows:

3.2.4 Conversion from Hexadecimal to Binary

Replace each hex digit by the 4 equivalent bits (as listed in the above table), for examples,

A3C5H = 1010 0011 1100 0101B

102AH = 0001 0000 0010 1010B

3.2.5 Conversion from Binary to Hexadecimal

Starting from the right-most bit (least-significant bit), replace each group of 4 bits by the equivalent hex digit (pad the left-most bits with zero if necessary), for examples, 1001001010B = 0010 0100 1010B = 24AH

10001011001011B = 0010 0010 1100 1011B = 22CBH

It is important to note that hexadecimal number provides a *compact form* or *shorthand* for representing binary bits.

3.2.6 Conversion from Base r to Decimal (Base 10)

Given a *n*-digit base *r* number: $d_{n-1}d_{n-2}d_{n-3}...d_2d_1d_0$ (base r), the decimal equivalent is given by:

$d_{n-1}\times r^{n-1} + d_{n-2}\times r^{n-2} + ... + d_1\times r^1 + d_0\times r^0$

For examples,

A1C2H = 10×16^3 + 1×16^2 + 12×16^1 + 2 = 41410 (base 10)

10110B = 1×2^4 + 1×2^2 + 1×2^1 = 22 (base 10)

3.2.7 Conversion from Decimal (Base 10) to Base r

Use repeated division/remainder. For example,

To convert 261(base 10) to hexadecimal:

261/16 => quotient=16 remainder=5

16/16 => quotient=1 remainder=0

1/16 => quotient=0 remainder=1 (quotient=0 stop)

Hence, 261D = 105H (Collect the hex digits from the remainder in reverse order)

The above procedure is actually applicable to conversion between any 2 base systems. For example,

To convert 1023(base 4) to base 3:

1023(base 4)/3 => quotient=25D remainder=0

25D/3 => quotient=8D remainder=1

8D/3 => quotient=2D remainder=2

2D/3 => quotient=0 remainder=2 (quotient=0 stop)

Hence, 1023(base 4) = 2210(base 3)

3.2.8 Conversion between Two Number Systems with Fractional Part

1. Separate the integral and the fractional parts.

2. For the integral part, divide by the target radix repeatably, and collect the ramainder in reverse order.
3. For the fractional part, multiply the fractional part by the target radix repeatably, and collect the integral part in the same order.

Example 1: Decimal to Binary

Convert 18.6875D to binary

Integral Part = 18D

18/2 => quotient=9 remainder=0

9/2 => quotient=4 remainder=1

4/2 => quotient=2 remainder=0

2/2 => quotient=1 remainder=0

1/2 => quotient=0 remainder=1 (quotient=0 stop)

Hence, 18D = 10010B

Fractional Part = .6875D

.6875*2=1.375 => whole number is 1

.375*2=0.75 => whole number is 0

.75*2=1.5 => whole number is 1

.5*2=1.0 => whole number is 1

Hence .6875D = .1011B

Combine, 18.6875D = 10010.1011B

Example 2: Decimal to Hexadecimal

Convert 18.6875D to hexadecimal

Integral Part = 18D

18/16 => quotient=1 remainder=2

1/16 => quotient=0 remainder=1 (quotient=0 stop)

Hence, 18D = 12H

Fractional Part = .6875D

.6875*16=11.0 => whole number is 11D (BH)

Hence .6875D = .BH

Combine, 18.6875D = 12.BH

3.3. Computer Memory & Data Representation:

Computer uses *a fixed number of bits* to represent a piece of data, which could be a number, a character, or others. A *n*-bit storage location can represent up to `2^`*n* distinct entities. For example, a 3-bit memory location can hold one of these eight binary patterns: `000`, `001`, `010`, `011`, `100`, `101`, `110`, or `111`. Hence, it can represent at most 8 distinct entities. You could use them to represent numbers 0 to 7, numbers 8881 to 8888, characters 'A' to 'H', or up to 8 kinds of fruits like apple, orange, banana; or up to 8 kinds of animals like lion, tiger, etc.

Integers, for example, can be represented in 8-bit, 16-bit, 32-bit or 64-bit. You, as the programmer, choose an appropriate bit-length for your integers. Your choice will impose constraint on the range of integers that can be represented. Besides the bit-length, an integer can be represented in various *representation* schemes, e.g., unsigned vs. signed integers. An 8-bit unsigned integer has a range of 0 to 255, while an 8-bit signed integer has a range of -128 to 127 - both representing 256 distinct numbers.

It is important to note that a computer memory location merely *stores a binary pattern*. It is entirely up to you, as the programmer, to decide on how these patterns are to be *interpreted*. For example, the 8-bit binary pattern `"0100 0001B"` can be interpreted as an unsigned integer `65`, or an ASCII character `'A'`, or some secret information known only to you. In other words, you have to first decide how to represent a piece of data in a binary pattern before the binary patterns make sense. The interpretation of binary pattern is called *data representation* or *encoding*. Furthermore, it is important that the data representation schemes are agreed-upon by all the parties, i.e., industrial standards need to be formulated and straightly followed.

Once you decided on the data representation scheme, certain constraints, in particular, the precision and range will be imposed. Hence, it is important to understand *data representation* to write *correct* and *high-performance* programs.

3.4. Integer Representation:

Integers are *whole numbers* or *fixed-point numbers* with the radix point *fixed* after the least-significant bit. They are contrast to *real numbers* or *floating-point numbers*, where the position of the radix point varies. It is important to take note that integers and floating-point numbers are treated differently in computers. They have different representation and are processed differently (e.g., floating-point numbers are processed in a so-called floating-point processor). Floating-point numbers will be discussed later.

Computers use *a fixed number of bits* to represent an integer. The commonly-used bit-lengths for integers are 8-bit, 16-bit, 32-bit or 64-bit. Besides bit-lengths, there are two representation schemes for integers:

1. *Unsigned Integers*: can represent zero and positive integers.
2. *Signed Integers*: can represent zero, positive and negative integers. Three representation schemes had been proposed for signed integers:
    1. Sign-Magnitude representation
    2. 1's Complement representation
    3. 2's Complement representation

You, as the programmer, need to decide on the bit-length and representation scheme for your integers, depending on your application's requirements. Suppose that you need a counter for counting a small quantity from 0 up to 200, you might choose the 8-bit unsigned integer scheme as there is no negative numbers involved.

3.4.1 n-bit Unsigned Integers

Unsigned integers can represent zero and positive integers, but not negative integers. The value of an unsigned integer is interpreted as "*the magnitude of its underlying binary pattern*".

Example 1: Suppose that *n*=8 and the binary pattern is `0100 0001B`, the value of this unsigned integer is `1×2^0 + 1×2^6 = 65D`.

Example 2: Suppose that *n*=16 and the binary pattern is `0001 0000 0000 1000B`, the value of this unsigned integer is `1×2^3 + 1×2^12 = 4104D`.

Example 3: Suppose that *n*=16 and the binary pattern is `0000 0000 0000 0000B`, the value of this unsigned integer is `0`.

An *n*-bit pattern can represent `2^n` distinct integers. An *n*-bit unsigned integer can represent integers from `0` to `(2^n)-1`, as tabulated below:

n	Minimum	Maximum
8	0	(2^8)-1 (=255)
16	0	(2^16)-1 (=65,535)
32	0	(2^32)-1 (=4,294,967,295) (9+ digits)
64	**0**	**(2^64)-1 (=18,446,744,073,709,551,615) (19+ digits)**

3.4.2 Signed Integers

Signed integers can represent zero, positive integers, as well as negative integers. Three representation schemes are available for signed integers:

1. Sign-Magnitude representation
2. 1's Complement representation
3. 2's Complement representation

In all the above three schemes, the *most-significant bit* (msb) is called the *sign bit*. The sign bit is used to represent the *sign* of the integer - with 0 for positive integers and 1 for negative integers. The *magnitude* of the integer, however, is interpreted differently in different schemes.

3.4.3 n-bit Sign Integers in Sign-Magnitude Representation

In sign-magnitude representation:

- The most-significant bit (msb) is the *sign bit*, with value of 0 representing positive integer and 1 representing negative integer.
- The remaining *n*-1 bits represents the magnitude (absolute value) of the integer. The absolute value of the integer is interpreted as "the magnitude of the (*n*-1)-bit binary pattern".

Example 1: Suppose that *n*=8 and the binary representation `0 100 0001B`.

Sign bit is `0` ⇒ positive

Absolute value is `100 0001B = 65D`

Hence, the integer is `+65D`

Example 2: Suppose that *n*=8 and the binary representation `1 000 0001B`.

Sign bit is `1` ⇒ negative

Absolute value is the complement of `000 0001B` plus `1`, i.e., `111 1110B + 1B = 127D`

Hence, the integer is `-127D`

Example 3: Suppose that *n*=8 and the binary representation `0 000 0000B`.

Sign bit is `0` ⇒ positive

Absolute value is `000 0000B = 0D`

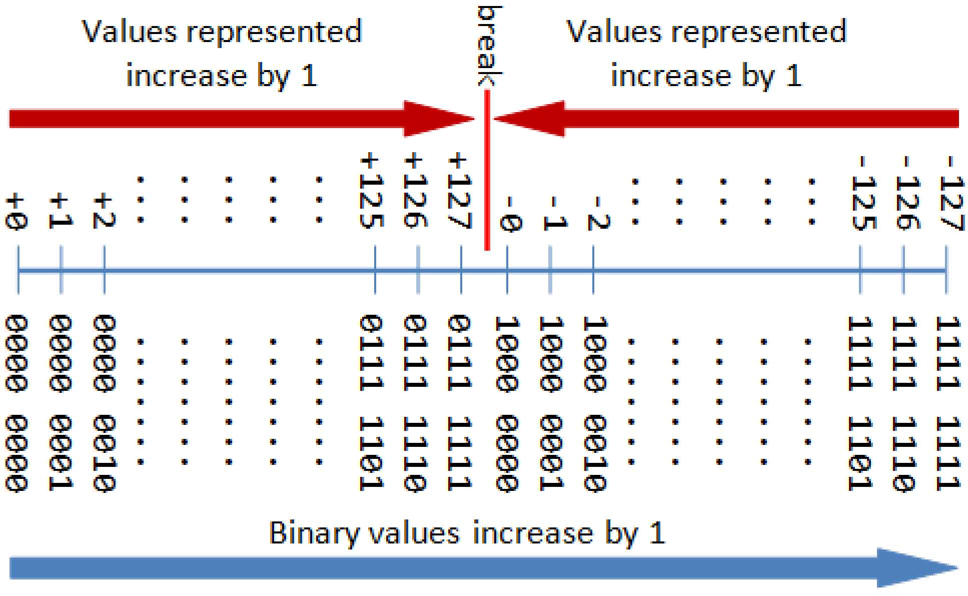
Hence, the integer is `+0D`

Example 4: Suppose that *n*=8 and the binary representation `1 111 1111B`.

Sign bit is `1` ⇒ negative

Absolute value is the complement of `111 1111B` plus `1`, i.e., `000 0000B + 1B = 1D`

Hence, the integer is `-1D`



Sign-Magnitude Representation

The drawbacks of sign-magnitude representation are:

1. There are two representations (0000 0000B and 1000 0000B) for the number zero, which could lead to inefficiency and confusion.
2. Positive and negative integers need to be processed separately.

3.4.4 n-bit Sign Integers in 1's Complement Representation

In 1's complement representation:

- Again, the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining *n*-1 bits represents the magnitude of the integer, as follows:
  - ✓ for positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".
  - ✓ for negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* (*inverse*) of the (*n*-1)-bit binary pattern" (hence called 1's complement).

Example 1: Suppose that *n*=8 and the binary representation `0 100 0001B`.

Sign bit is `0` ⇒ positive

Absolute value is `100 0001B = 65D`

Hence, the integer is `+65D`

Example 2: Suppose that *n*=8 and the binary representation `1 000 0001B`.

Sign bit is `1` ⇒ negative

Absolute value is the complement of `000 0001B`, i.e., `111 1110B = 126D`

Hence, the integer is `-126D`

Example 3: Suppose that *n*=8 and the binary representation `0 000 0000B`.

Sign bit is `0` ⇒ positive

Absolute value is `000 0000B = 0D`

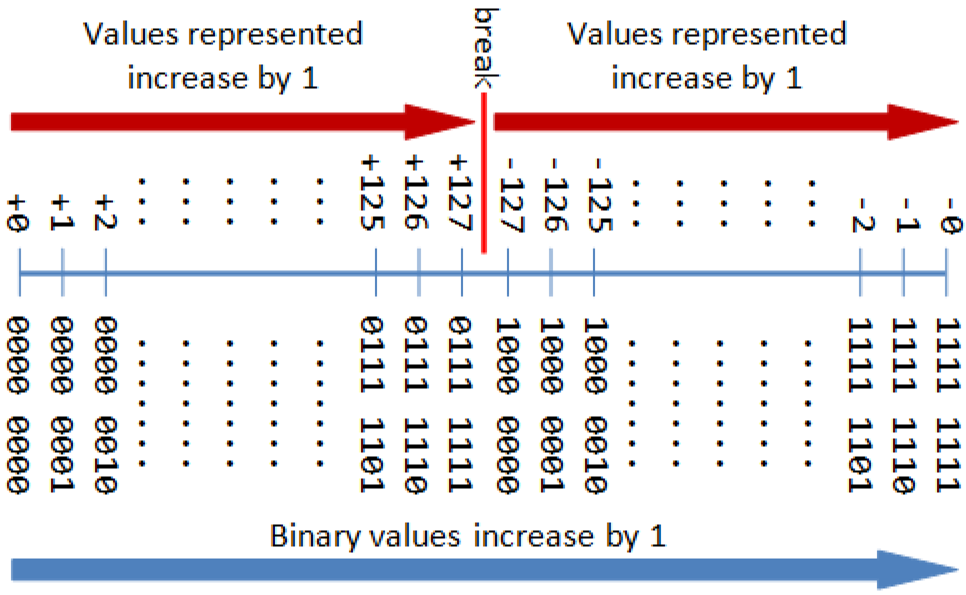
Hence, the integer is `+0D`

Example 4: Suppose that *n*=8 and the binary representation `1 111 1111B`.

Sign bit is `1` ⇒ negative

Absolute value is the complement of `111 1111B`, i.e., `000 0000B = 0D`

Hence, the integer is `-0D`



1's Complement Representation

Again, the drawbacks are:

1. There are two representations (0000 0000B and 1111 1111B) for zero.
2. The positive integers and negative integers need to be processed separately.

3.4.5 n-bit Sign Integers in 2's Complement Representation

In 2's complement representation:

- Again, the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining *n*-1 bits represents the magnitude of the integer, as follows:
  - ✓ for positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".

- ✓ for negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* of the (n-1)-bit binary pattern *plus one*" (hence called 2's complement).

Example 1: Suppose that n=8 and the binary representation `0 100 0001B`.

Sign bit is `0` ⇒ positive

Absolute value is `100 0001B = 65D`

Hence, the integer is `+65D`

Example 2: Suppose that n=8 and the binary representation `1 000 0001B`.

Sign bit is `1` ⇒ negative

Absolute value is the complement of `000 0001B` plus `1`, i.e., `111 1110B + 1B = 127D`

Hence, the integer is `-127D`

Example 3: Suppose that n=8 and the binary representation `0 000 0000B`.

Sign bit is `0` ⇒ positive

Absolute value is `000 0000B = 0D`

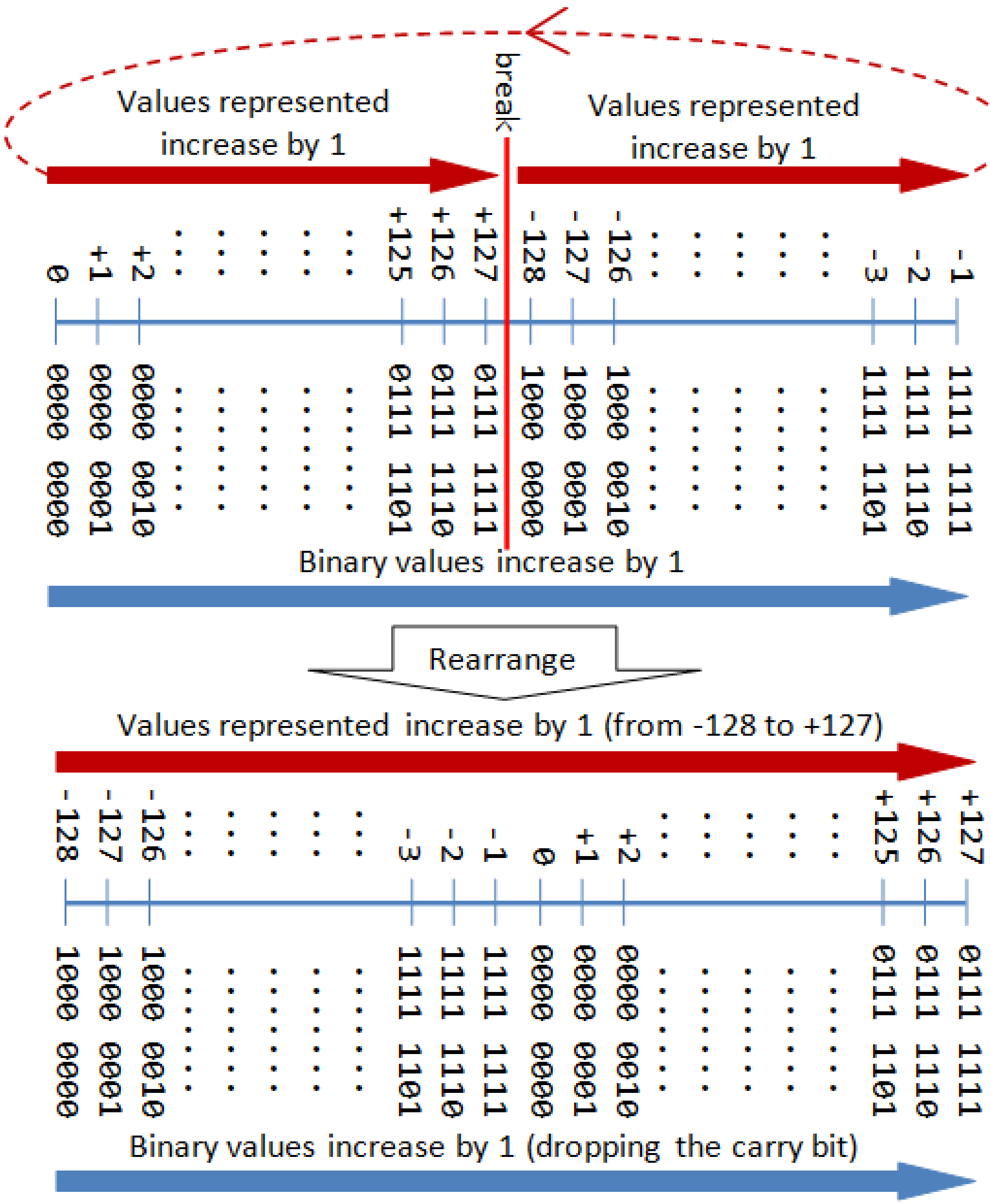
Hence, the integer is `+0D`

Example 4: Suppose that n=8 and the binary representation `1 111 1111B`.

Sign bit is `1` ⇒ negative

Absolute value is the complement of `111 1111B` plus `1`, i.e., `000 0000B + 1B = 1D`

Hence, the integer is `-1D`



2's Complement Representation

3.4.6 Computers use 2's Complement Representation for Signed Integers

We have discussed three representations for signed integers: signed-magnitude, 1's complement and 2's complement. Computers use 2's complement in representing signed integers. This is because:

1. There is only one representation for the number zero in 2's complement, instead of two representations in sign-magnitude and 1's complement.
2. Positive and negative integers can be treated together in addition and subtraction. Subtraction can be carried out using the "addition logic".

Example 1: Addition of Two Positive Integers: Suppose that `n=8, 65D + 5D = 70D`

65D → 0100 0001B

5D → 0000 0101B(+

0100 0110B → 70D (OK)

Example 2: Subtraction is treated as Addition of a Positive and a Negative Integers: Suppose that `n=8, 5D - 5D = 65D + (-5D) = 60D`

65D → 0100 0001B

-5D → 1111 1011B(+

0011 1100B → 60D (discard carry - OK)

Example 3: Addition of Two Negative Integers: Suppose that `n=8, -65D - 5D = (-65D) + (-5D) = -70D`

-65D → 1011 1111B

-5D → 1111 1011B(+

1011 1010B → -70D (discard carry - OK)

Because of the *fixed precision* (i.e., *fixed number of bits*), an *n*-bit 2's complement signed integer has a certain range. For example, for *n*=8, the range of 2's complement signed integers is `-128` to `+127`. During addition (and subtraction), it is important to check whether the result exceeds this range, in other words, whether *overflow* or *underflow* has occurred.

Example 4: Overflow: Suppose that `n=8, 127D + 2D = 129D` (overflow - beyond the range)

127D → 0111 1111B

2D → 0000 0010B(+

1000 0001B → -127D (wrong)

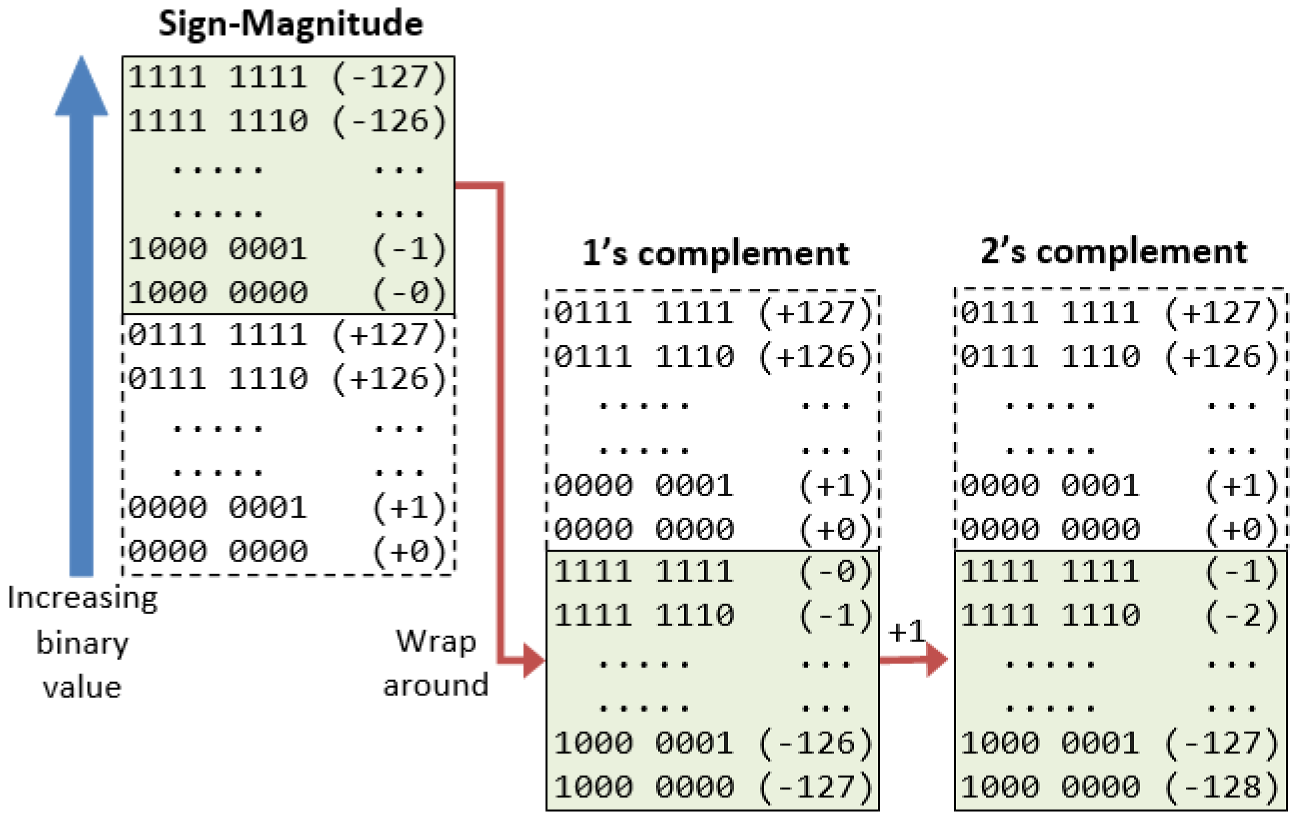
Example 5: Underflow: Suppose that `n=8, -125D - 5D = -130D` (underflow - below the range)

-125D → 1000 0011B

-5D → 1111 1011B(+

0111 1110B → +126D (wrong)

The following diagram explains how the 2's complement works. By re-arranging the number line, values from `-128` to `+127` are represented contiguously by ignoring the carry bit.



3.4.7 Range of n-bit 2's Complement Signed Integers

An *n*-bit 2's complement signed integer can represent integers from `-2^(n-1)` to `+2^(n-1)-1`, as tabulated. Take note that the scheme can represent all the integers within the range, without any gap. In other words, there is no missing integers within the supported range.

n	minimum	maximum
8	-(2^7) (=-128)	+(2^7)-1 (=+127)
16	-(2^15) (=-32,768)	+(2^15)-1 (=+32,767)
32	-(2^31) (=-2,147,483,648)	+(2^31)-1 (=+2,147,483,647)(9+ digits)
64	-(2^63) (=-9,223,372,036,854,775,808)	+(2^63)-1 (=+9,223,372,036,854,775,807)(18+ dig

3.4.8 Decoding 2's Complement Numbers

1. Check the *sign bit* (denoted as S).
2. If S=0, the number is positive and its absolute value is the binary value of the remaining *n*-1 bits.
3. If S=1, the number is negative. you could "invert the *n*-1 bits and plus 1" to get the absolute value of negative number. Alternatively, you could scan the remaining *n*-1 bits from the right (least-significant

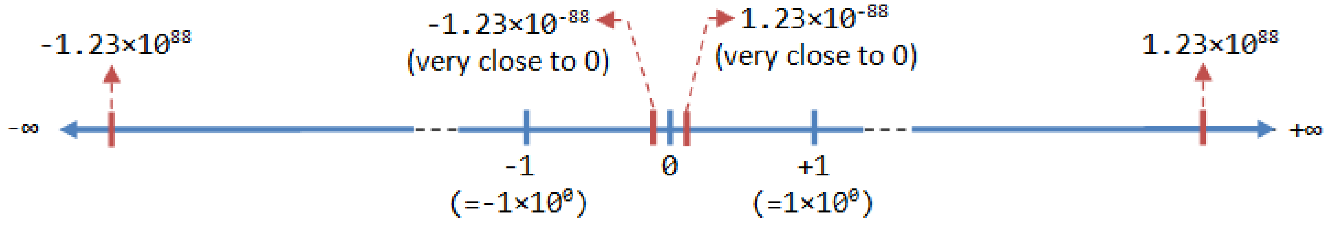
bit). Look for the first occurrence of 1. Flip all the bits to the left of that first occurrence of 1. The flipped pattern gives the absolute value. For example,

4. n = 8, bit pattern = 1 100 0100B
5. S = 1 → negative
6. Scanning from the right and flip all the bits to the left of the first occurrence of 1 ⇒011 1100B = 60D

Hence, the value is -60D

3.5. Floating-Point Number Representation:

A floating-point number (or real number) can represent a very large (`1.23×10^88`) or a very small (`1.23×10^-88`) value. It could also represent very large negative number (`-1.23×10^88`) and very small negative number (`-1.23×10^88`), as well as zero, as illustrated:



Floating-point Numbers (Decimal)

A floating-point number is typically expressed in the scientific notation, with a *fraction* (`F`), and an *exponent* (`E`) of a certain *radix* (`r`), in the form of `F×r^E`. Decimal numbers use radix of 10 (`F×10^E`); while binary numbers use radix of 2 (`F×2^E`).

Representation of floating point number is not unique. For example, the number 55.66 can be represented as 5.566×10^1, 0.5566×10^2, 0.05566×10^3, and so on. The fractional part can be *normalized*. In the normalized form, there is only a single non-zero digit before the radix point. For example, decimal number 123.4567 can be normalized as 1.234567×10^2; binary number 1010.1011B can be normalized as 1.0101011B×2^3.

It is important to note that floating-point numbers suffer from *loss of precision* when represented with a fixed number of bits (e.g., 32-bit or 64-bit). This is because there are *infinite* number of real numbers (even within a small range of says 0.0 to 0.1). On the other hand, a *n*-bit binary pattern can represent a *finite* 2^*n* distinct numbers. Hence, not all the real numbers can be represented. The nearest approximation will be used instead, resulted in loss of accuracy.

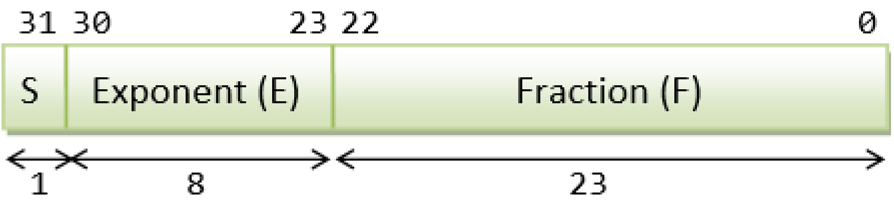
It is also important to note that floating number arithmetic is very much less efficient than integer arithmetic. It could be speed up with a so-called dedicated *floating-point co-processor*. Hence, use integers if your application does not require floating-point numbers.

In computers, floating-point numbers are represented in scientific notation of *fraction* (F) and *exponent* (E) with a *radix* of 2, in the form of F×2^E. Both E and F can be positive as well as negative. Modern computers adopt IEEE 754 standard for representing floating-point numbers. There are two representation schemes: 32-bit single-precision and 64-bit double-precision.

3.5.1 IEEE-754 32-bit Single-Precision Floating-Point Numbers

In 32-bit single-precision floating-point representation:

- The most significant bit is the *sign bit* (S), with 0 for positive numbers and 1 for negative numbers.
- The following 8 bits represent *exponent* (E).
- The remaining 23 bits represents *fraction* (F).



32-bit Single-Precision Floating-point Number

Normalized Form

Let's illustrate with an example, suppose that the 32-bit pattern is 1 1000 0001 011 0000 0000 0000 0000 0000, with:

- S = 1
- E = 1000 0001

➢ F = 011 0000 0000 0000 0000 0000

In the *normalized form*, the actual fraction is normalized with an implicit leading 1 in the form of `1.F`. In this example, the actual fraction is `1.011 0000 0000 0000 0000 0000 = 1 + 1×2^-2 + 1×2^-3 = 1.375D`.

The sign bit represents the sign of the number, with `S=0` for positive and `S=1` for negative number. In this example with `S=1`, this is a negative number, i.e., `-1.375D`.

In normalized form, the actual exponent is `E-127` (so-called excess-127 or bias-127). This is because we need to represent both positive and negative exponent. With an 8-bit E, ranging from 0 to 255, the excess-127 scheme could provide actual exponent of -127 to 128. In this example, `E-127=129-127=2D`.

Hence, the number represented is `-1.375×2^2=-5.5D`.

De-Normalized Form

Normalized form has a serious problem, with an implicit leading 1 for the fraction, it cannot represent the number zero! Convince yourself on this!

De-normalized form was devised to represent zero and other numbers.

For `E=0`, the numbers are in the de-normalized form. An implicit leading 0 (instead of 1) is used for the fraction; and the actual exponent is always `-126`. Hence, the number zero can be represented with `E=0` and `F=0` (because `0.0×2^-126=0`).

We can also represent very small positive and negative numbers in de-normalized form with `E=0`. For example, if `S=1`, `E=0`, and `F=011 0000 0000 0000 0000 0000`. The actual fraction is `0.011=1×2^-2+1×2^-3=0.375D`. Since `S=1`, it is a negative number. With `E=0`, the actual exponent is `-126`. Hence the number is `-0.375×2^-126 = -4.4×10^-39`, which is an extremely small negative number (close to zero).

Summary

In summary, the value (`N`) is calculated as follows:

- For $1 \le E \le 254$, N = (-1)^S × 1.F × 2^(E-127). These numbers are in the so-called *normalized* form. The sign-bit represents the sign of the number. Fractional part (1.F) are normalized with an implicit leading 1. The exponent is bias (or in excess) of 127, so as to represent both positive and negative exponent. The range of exponent is -126 to +127.
- For E = 0, N = (-1)^S × 0.F × 2^(-126). These numbers are in the so-called *denormalized* form. The exponent of 2^-126 evaluates to a very small number.

Denormalized form is needed to represent zero (with F=0 and E=0). It can also represents very small positive and negative number close to zero.

- For E = 255, it represents special values, such as ±INF (positive and negative infinity) and NaN (not a number). This is beyond the scope of this article.

Example 1: Suppose that IEEE-754 32-bit floating-point representation pattern is 0 10000000 110 0000 0000 0000 0000 0000.

Sign bit S = 0 ⇒ positive number

E = 1000 0000B = 128D (in normalized form)

Fraction is 1.11B (with an implicit leading 1) = 1 + 1×2^-1 + 1×2^-2 = 1.75D

The number is +1.75 × 2^(128-127) = +3.5D

Example 2: Suppose that IEEE-754 32-bit floating-point representation pattern is 1 01111110 100 0000 0000 0000 0000 0000.

Sign bit S = 1 ⇒ negative number

E = 0111 1110B = 126D (in normalized form)

Fraction is 1.1B (with an implicit leading 1) = 1 + 2^-1 = 1.5D

The number is -1.5 × 2^(126-127) = -0.75D

Example 3: Suppose that IEEE-754 32-bit floating-point representation pattern is 1 01111110 000 0000 0000 0000 0000 0001.

Sign bit S = 1 ⇒ negative number

E = 0111 1110B = 126D (in normalized form)

Fraction is 1.000 0000 0000 0000 0000 0001B (with an implicit leading 1) = 1 + 2^-23

The number is -(1 + 2^-23) × 2^(126-127) = -0.500000059604644775390625 (may not be exact in decimal!)

Example 4 (De-Normalized Form): Suppose that IEEE-754 32-bit floating-point representation pattern is 1 00000000 000 0000 0000 0000 0000 0001.

Sign bit S = 1 ⇒ negative number

E = 0 (in de-normalized form)

Fraction is 0.000 0000 0000 0000 0000 0001B (with an implicit leading 0) = 1×2^-23

The number is $-2^{-23} \times 2^{(-126)} = -2\times(-149) \approx -1.4\times10^{-45}$

UNIT – 03 – Part - B

COMPUTER ARITHMETIC

Arithmetic instructions in digital computers manipulate data to produce results necessary for the solution of computational problems. These instructions perform arithmetic calculations and are responsible for the bulk of activity involved in processing data in a computer. The four basic arithmetic operations are addition, subtraction, multiplication and division. From these four basic operations, it is possible to formulate other arithmetic functions and solve scientific problems by means of numerical analysis methods. An arithmetic processor is the part of a processor unit that executes arithmetic operations. An arithmetic instruction mayspecifybinaryordecimaldata,andineachcasethedatamaybeinfixed-pointorfloating-pointform.Fixed-pointnumbersmayrepresentintegersorfractions.Negativenumbersmaybeinsigned-magnitudeor signed-complement representation. The arithmetic processor is very simple if only a binary fixed-point odd instruction is included. It would be more complicated if it includes all four arithmetic operations for binary and decimal data in fixed-point and floating-point representation.

3.6 Addition and Subtraction:

There are three ways of representing negative fixed-point binary numbers: signed-magnitude, signed-l's complement, or signed-2's complement. Most computers use the signed-2'scomplement representation when performing arithmetic operations with integers. For floating- point operations, most computers use the signed-magnitude representation for the mantissa. In this section we develop the addition and subtraction algorithms for data represented in signed-magnitude and again for data represented in signed-2's complement.

Addition and Subtraction with Signed-Magnitude Data

The representation of numbers in signed-magnitude is familiar because it is used in everyday arithmetic calculations. The procedure for adding or subtracting two signed binary numbers with paper and pencil Is simple and straight-forward. We designate the magnitude of the two numbers by A and B. When the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the first column of Table. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to prevent a negative zero. In other words, when two equal numbers are subtracted, the result should be +0not–0.

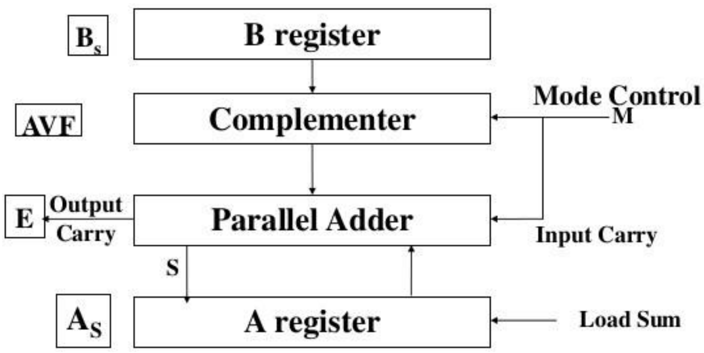
Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

The algorithms for addition and subtraction are derived from the table and can be stated as follows (thewordsinsideparenthesesshouldbeusedforthesubtractionalgorithm):

Addition (subtraction)algorithm: when the signs of A and B are identical (different),add the two magnitudes and attach the sign of A to the result. When the signs of A and B are different (identical),compare the magnitudes and subtract the smaller number from the larger. Choose the sign of the result to be the same as A if A > B or the complement of the sign of A if A < B. If the two magnitudes are equal, subtract B from A and make the sign of the result positive. The two algorithms are similar except for thesigncomparison.Theproceduretobefollowedforidenticalsignsintheadditionalgorithmisthesameasfordifferentsigns in the subtraction algorithm, and vice versa.

Hardware Implementation

To implement the two arithmetic operations with hardware, it is first necessary that the two numbers bestoredinregisters.LetAandBbetworegistersthatholdthemagnitudesofthenumbers,andAsandBs be two flip-flops that hold the corresponding signs. The result of the operation may be transferred to a thirdregister: however, a saving is achieved if the result is transferred into A and As. Thus A and As togetherform an accumulator register. Consider nowthe hardware implementation of the algorithms above. First, aparallel-adder is needed toperform the micro-operation A + B. Second, a comparator circuit is needed toestablish if A > B, A = B, or A < B. Third, two parallel-subtractor circuits are needed to perform themicro- operations A - B and B - A. The sign relationship can be determined from an exclusive-OR gate withA s and B s as inputs. This procedure requires a magnitude comparator, an adder, and two subtractors.However, a different procedure can be found that requires less equipment. First, we know that subtractioncan be accomplished by means of complement andadd.Second,theresult of a comparison can bedetermined from the end carry after the subtraction. Careful investigation of the alternatives reveals that theuse of 2's complement for subtraction and comparison is an efficient procedure that requires only an adderand a complementor. Figure shows a block diagram of the hardware for implementing the addition andsubtractionoperations.

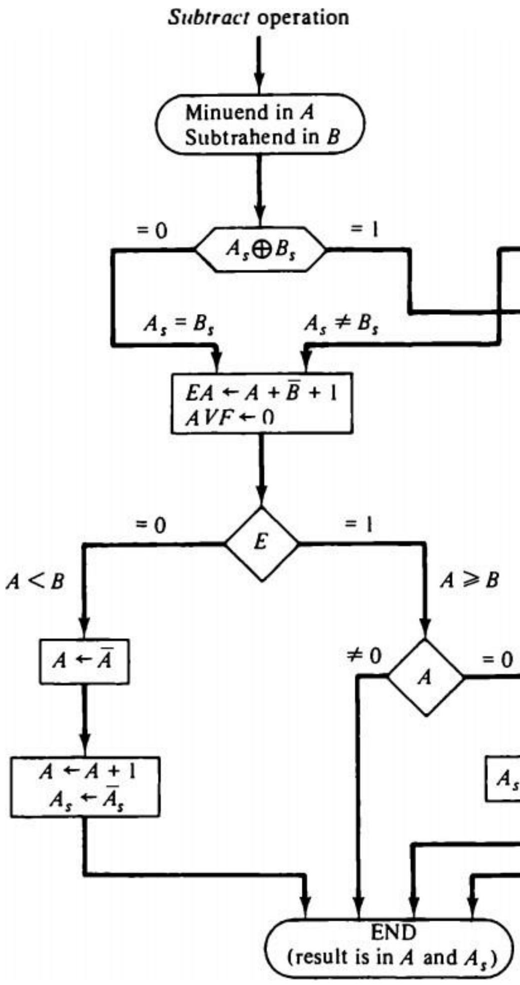


It consists of registers A and B and sign flip-flops Asand Bs. Subtraction is done by adding A tothe 2'scomplement of B. The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers. The add-overflow flip-flop AVF holds the overflow bit when A and B are added. The A register provides other micro-operations that may be needed when we specify the sequence of steps in the algorithm.

The addition of A plus B is done through the parallel adder. The S(sum) output of the adder is applied to the input of the A register. The complementor provides an output of B or the complement of B depending on the state of the mode control M. The complementor consists of exclusive-OR gates and the parallel adder consists of full-adder circuits. The M signal is also applied to the input carry of the adder. When M = 0, the output of B is transferred to the adder, the input carry is 0, and the output of the adder is equal to the sum A+B. When M = 1, the 2's complement of B is applied to the adder the input carry is 1, and output S = A +B'—1.Thisis equal to A plus the2'scomplement of B, which is equivalent to the subtraction A—B.

Hardware Algorithm

The flow chart for the hardware algorithm is presented in Figure. The two signs As, and Bs are compared by an exclusive-OR gale. If the output of the gate is 0, the signs are identical; if it is 1, the signs are different for an add operation, identical signs dictate that



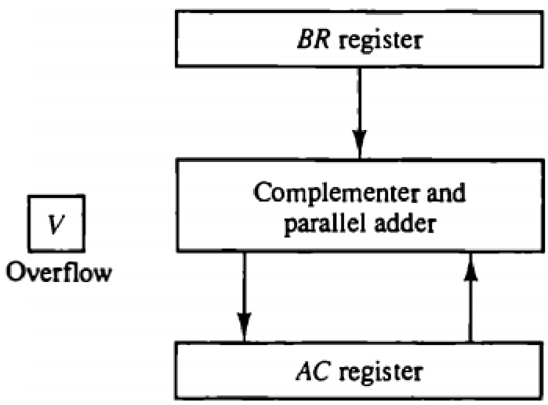
the magnitudes be added.

For a subtract operation, different signs dictate that the magnitudes be added. The magnitudes are added with a micro-operation EA A +B where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1 and it is transferred into the add-overflow flip-flop AVF. The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complement of B. No overflow can occur if the numbers are subtracted. A 1 in E indicates that A ≥ B and the number in A is the correctresult. If this number is zero, the sign As must be made positive to avoid a –0. A 0 in E indicates that A < B.For this case it is necessary to take the 2's complement o fthe value in A. This operation can be done with one micro-operation A A' + 1. In other paths of the flowchart, the sign of the result is the same asthe sign of A, sono change in A_S is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is the necessary to complement A_S to obtain the correct sign. The final result is found in register A and its sign in A_S.The value in AVF provides an overflowindication. The final value of E is immaterial.

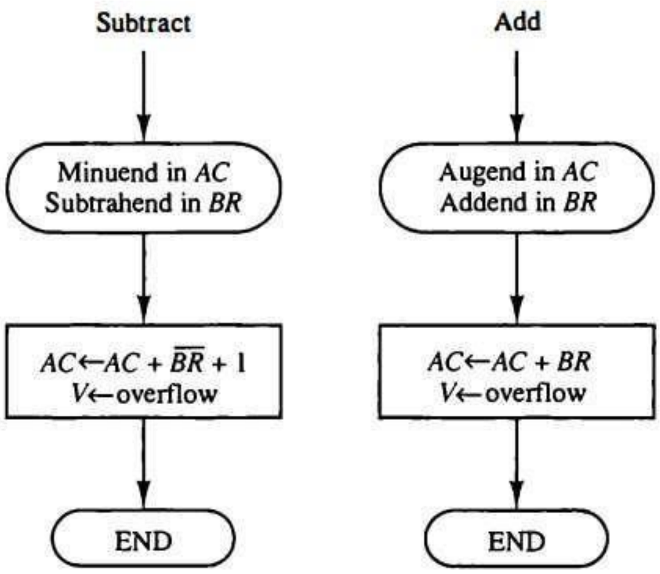
AdditionandSubtractionwithSigned-2'sComplementData

The signed-2's complement representation of numbers together with arithmetic algorithms for addition and subtraction are summarized here for easy reference. The leftmost bit of a binary number represents the sign bit: 0 for positive and 1 for negative. If the sign bit is 1, the entire number is represented in 2's complement form. Thus +33 is represented as 00100001 and -33 as 11011111. Note that 11011111 is the 2's complementof00100001,and vice versa.

The addition of two numbers in signed-2's complement form consists of adding the numbers with the sign bits treated the same as the other bits of the number. A carry-out of the sign-bit position is discarded. The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend. When two numbers of *n* digits each are added and the sum occupies n + 1 digits, we say that an overflow occurred. An over flow can be detected by inspecting the last two carries out of the addition. When the two carries are applied to an exclusive-OR gate, the overflow is detected when the output of the gate is equal to 1. The register configuration for the hardware implementation is shown in Figure. The sign bits are not separated from the rest of the registers. We name the A register AC and the B register BR. The left most bit in AC and BR represent the sign bits of the numbers. The two sign bits are added or subtracted together with the other bits in the complementor and parallel adder. The overflow flip-flop V is set to 1 if there is an overflow. The output carry in this case is discarded.



The algorithm for adding and subtracting two binary numbers in signed-2's complement representation is shown in the flowchart of Figure. The sum is obtained by adding the contents of AC and BR (including their sign bits). The overflow bit V is set to 1 if the exclusive-OR of the last two carries is 1,and it is cleared to 0 otherwise. The subtraction operation is accomplished by adding the content of AC to the 2'scomplement of BR. Taking the 2's complement of BR has the effect of changing a positive number to negative, and vice versa. An overflowmustbechecked during this operation because the two numbersadded could have the same sign. The programmer must realize that if an overflow occurs, there will be anerroneousresultin theAC register.



Comparing this algorithm with its signed-magnitude counterpart, we note that itismuchsimpler to addand subtract numbers if negative numbers are maintained in signed-2's complement representation. For thisreasonmostcomputers adoptthisrepresentation overthemorefamiliarsigned-magnitude.

3.7. MultiplicationAlgorithm

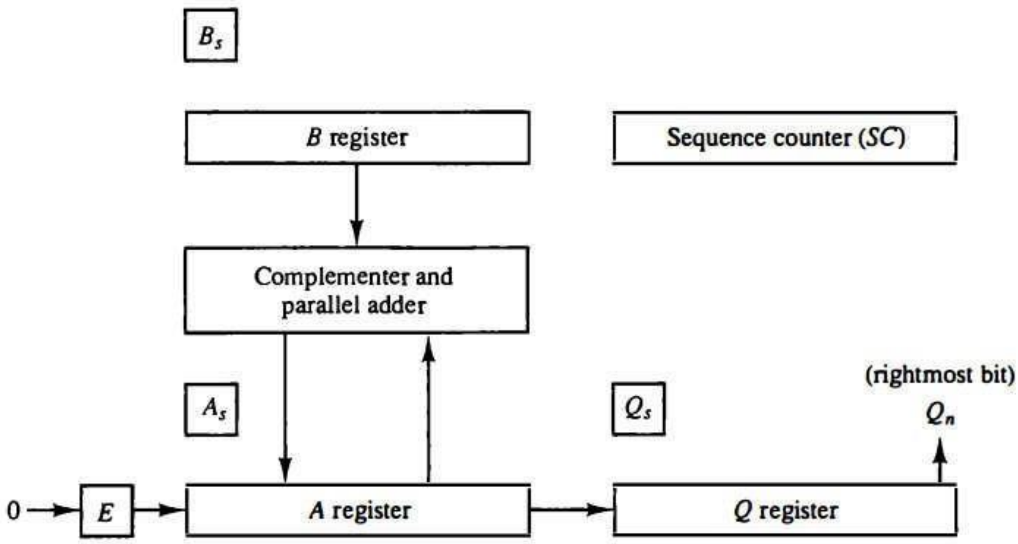
Multiplication of two fixed-point binary numbers in signed-magnitude representation is done withpaper and pencil by a process of successive shift and add operations. This process is best illustratedwithanumericalexample.

```
 23        10111    Multiplicand
 19      × 10011    Multiplier
           10111
          10111
         00000   +
        00000
       10111
437    110110101    Product
```

The process consists of looking at successive bits of the multiplier, least significant bit first. If themultiplier bit is a 1, the multiplicand is copied down; otherwise, zeros are copied down. The numberscopied down in successive lines are shifted one position to the left from the previous number. Finally,the numbers are added and their sum forms the product. The sign of the product is determined from thesigns of the multiplicand and multiplier. If they are alike, the sign of the product is positive. If they are unlike, the sign of the product is negative.

Hardware Implementation for Signed-Magnitude Data

When multiplication is implemented in a digital computer, it is convenient to change the processslightly.First,insteadofprovidingregisterstostoreandaddsimultaneouslyasmanybinarynumbersas there are bits in the multiplier, it is convenient to provide an adder for the summation of only two binary numbers and successively accumulate the partial products in a register. Second, instead of shifting the multiplicand to the left, the partial product is shifted to the right, which results in leavingthepartialproductandthemultiplicandintherequiredrelativepositions.Third,whenthecorresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product since it will not alter its value. The hardware for multiplication consists of the equipment shown in Figure. The multiplier is stored in the Q register and its sign in Qs. The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.

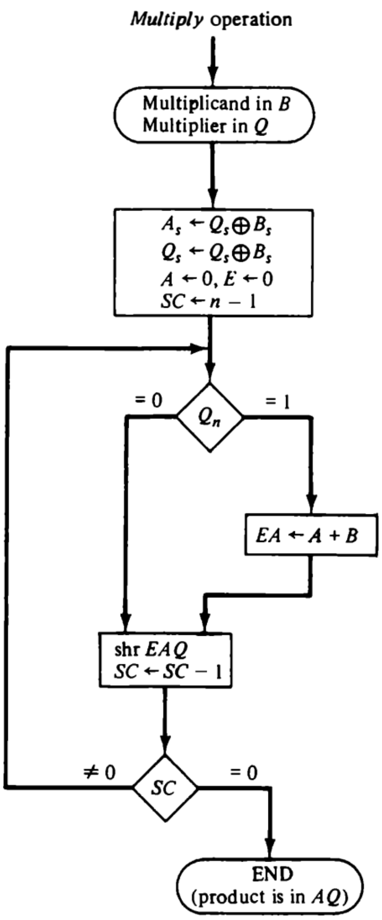


Initially, the multiplicand is in register B and the multiplier in Q. The sum of A and B forms a partial product which is transferred to the EA register. Both partial product and multiplier are shifted to the right. This shift will be denoted by the statement shr EAQ to designate the right shift depicted in Figure. The least significant bit of A is shifted into the most significant position of Q, the bit from E Is shifted into the most significant position of A, and 0 is shifted into E. After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right. In this manner, the rightmost flip-flop in register Q, designated by Q„, will hold the bit of the multiplier, which must be inspected next.

Hardware Algorithm

Figure shows a flowchart of the hardware multiply algorithm. Initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs, respectively. The signs are compared, and both A and Q are set to correspond to the sign of the product since a double- length product will be stored in registers A and Q. Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier. We are assuming here that operands are transferred to registers from a memory unit that has words of n bits. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n — 1bits.

After the initialization, the low-order bit of the multiplier in Qn is tested. If it is a 1, the multiplicand in B isadded to the present partial product in A. If it is a 0, nothingis done.Register



EAQ is then shifted once totherightto formthe newpartialproduct. Thesequencecounter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated anda new partial product is formed. The process stops when SC = 0. Note that the partial product formed in A isshiftedintoQonebitatatimeandeventuallyreplacesthemultiplier.Thefinalproductisavailableinbot hA and Q, with A holdingthemost significantbits and Q holding the least significant bits. The previousnumericalexampleisrepeatedin Table is shown to clarify the hardware multiplication process. Theprocedurefollows the stepsoutlined in theflowchart.

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Q_n = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Q_n = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Q_n = 0; shift right EAQ	0	01000	10110	010
Q_n = 0; shift right EAQ	0	00100	01011	001
Q_n = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 0110110101				

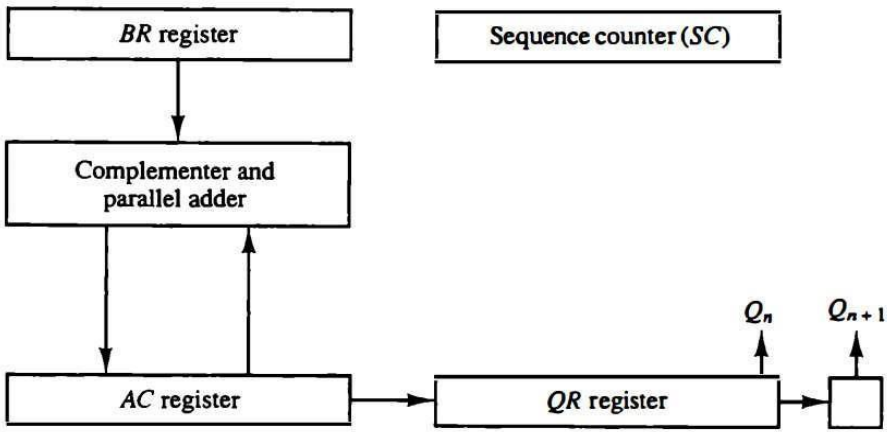
BoothMultiplicationAlgorithm

Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string ofTs in the multiplier from bit weight 2* to weight 2 m can betreatedas 2 k+1 — 2 m . For example, thebinary number 001110 (+14) has a string of 1's from2^k to2^m (k = 3, m = 1). The number can berepresented as 2^{k+1} - 2^{m} = 16 - 2 = 14. Therefore, the multiplicationMx14, whereMisthemultiplicandand14themultiplier,canbedoneasMx24M x 2^{1}. Thus the product can be obtained by shifting the binary multiplicandM fourtimes tothe left andsubtracting M shifted left once. As in all multiplication schemes. Booth algorithm requires examination ofthe multiplier bits and shifting of the partial product. Prior to theshifting, the multiplicand may be added tothepartialproduct, subtractedfromthe partialproduct,orleftunchangedaccording tothefollowingrules:

- Themultiplicandissubtractedfromthepartialproductuponencounteringthefirstleasts ignificant 1inastringof1's inthemultiplier.
- Themultiplicandisaddedtothepartialproductuponencounteringthefirst0(providedthatth erewasaprevious 1)ina string of0's inthemultiplier.

- Thepartialproductdoesnotchangewhenthemultiplierbitisidenticaltothepreviousmul tiplierbit.

The algorithm works for positive or negative multipliers in 2'scomplement representation. Thisis becausea negative multiplier ends with a string of 1's and the last operation will be asubtraction of the appropriateweight. For example,a multiplier equal to -14 is representedin2's complement as 110010 and is treatedas-2^4 +2^2 −2^1=−14.

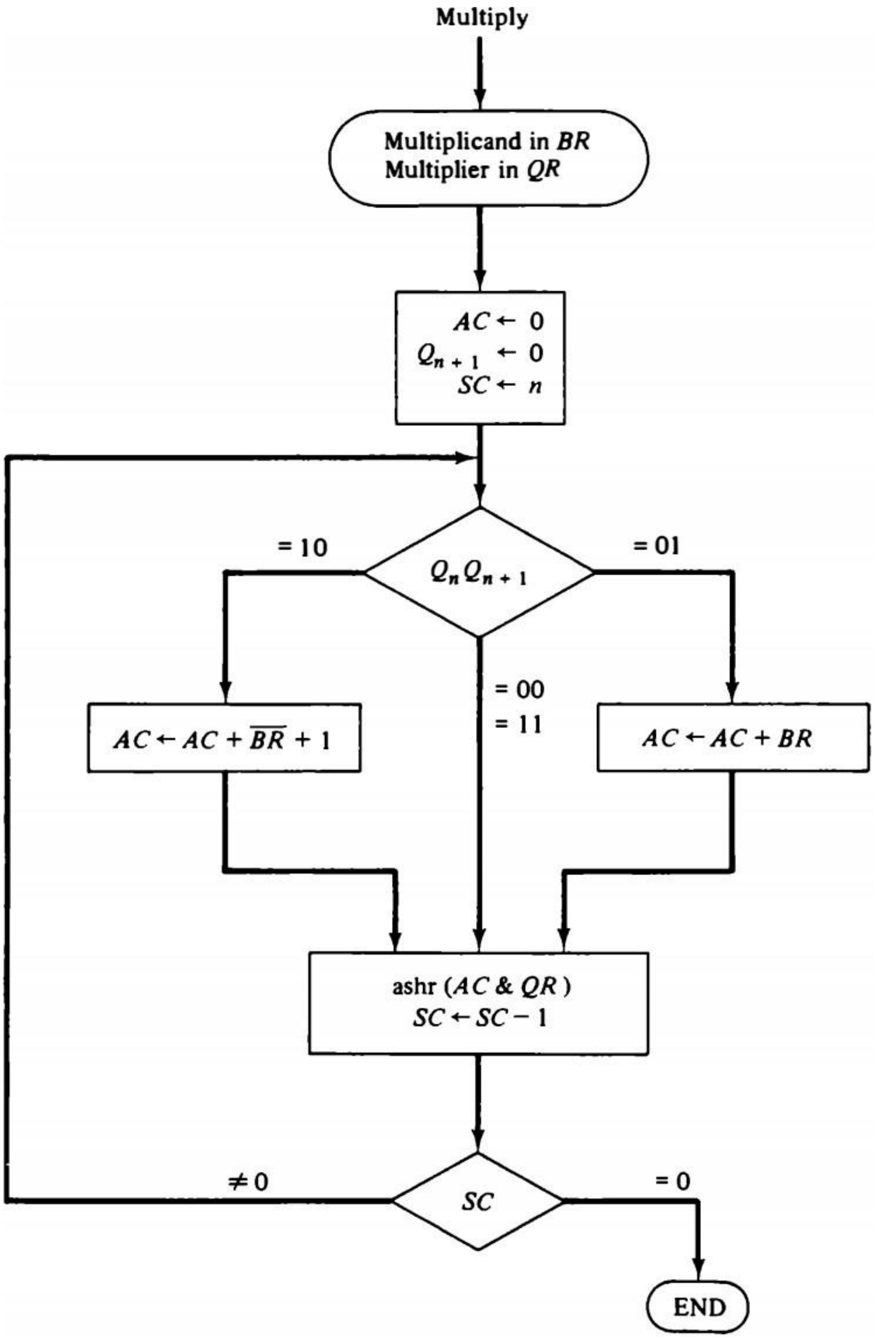


The hardware implementation of Booth algorithm requires the register configuration shown in Figure. Werename registers A, B, and Q, as AC, BR,andQR, respectively. Qndesignates theleast significant bit ofthe multiplier in register QR. An extra flip-flop Qn+1is appended to QR to facilitate a double bit inspectionof the multiplier. The flowchart for Booth algorithm is shown in Figure. AC and the appended bit Qn+1are initially cleared to 0 and the sequence counter SC is set to a number n equal to the number of bits in themultiplier. The two bits ofthe multiplier inQnand Qn+1are inspected. If the two bits are equal to 10, itmeans that the first 1 in a string of l's has been encountered. This requires a subtraction ofthe multiplicandfrom thepartialproductin AC. If the two bits are equal to 01, it means that the first 0 in a string of 0's hasbeen encountered. This requires the addition of the multiplicand to the partial product in AC.

Whenthe twobits are equal, the partial product does not change.An overflow cannot occurbecausethe addition andsubtraction of the multiplicand follow each other. As a consequence, the two numbers that are added alwayshave opposite signs, a condition that excludes an overflow. The next step is to shift right the partial productand the multiplier (including bit Qn+1). This is an arithmetic shift right (ashr) operation which shifts ACandQR to the right and leaves the signbit in AC unchanged. The sequence counter is decremented and thecomputationalloop is repeated ntimes.

Q_n Q_{n+1}	BR = 10111 $\overline{BR}$ + 1 = 01001	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

AnumericalexampleofBoothalgorithmisshowninTableforn=5.Itshowsthestep-by-stepmultiplication of (-9) x (-13) = +117. Note that the multiplier in QR is negative and that the multiplicand inBR is also negative. The 10-bit product appears in AC and QRandis positive.The final value of Qn+1 istheoriginalsign bitofthe multiplierand should notbe taken aspartofthe product.



3.8. Division Algorithms:

Division of two fixed-point binary numbers in signed-magnitude representation is done with paper andpencil by a process of successive compare, shift, and subtract operations. Binary division is simplerthan decimal division because the quotient digits are either 0 or 1 and there is no need to estimate howmany times the dividend or partial remainder fits into the divisor. The division process is illustrated bya numerical example in

Figure. The divisor B consists of five bits and the dividend A, of ten bits. Thefive most significant bits of the dividend are compared with the divisor. Since the 5-bit number issmallerthanB,wetryagainbytakingthesixmostsignificantbitsofAand compare thisnumberwithB.The6-bitnumberisgreaterthanB,soweplacea1forthequotientbitinthesixthpositionabovethedividend. The divisor is then shifted once to the right and subtracted from the dividend. The difference iscalled a partial remainder because the division could have stopped here to obtain a quotient of 1 and aremainder equal to the partial remainder. The process is continued by comparing a partial remainderwith the divisor. If the partial remainder is greater than or equal to the divisor, the quotient bit is equalto 1. The divisor is then shifted right and subtracted from the partial remainder. If the partial remainderis smaller than the divisor, the quotient bit is 0 and no subtraction is needed. The divisor is shifted oncetotheright inany case.Notethat theresult gives botha quotient andaremainder.

```
Divisor:              11010        Quotient = Q
B = 10001        )0111000000       Dividend = A
                  01110            5 bits of A < B, quotient has 5 bits
                  011100           6 bits of A > B
                  -10001           Shift right B and subtract; enter 1 in Q
                  -010110          7 bits of remainder > B
                  --10001          Shift right B and subtract; enter 1 in Q
                  --001010         Remainder < B; enter 0 in Q; shift right B
                  ---010100        Remainder > B
                  ----10001        Shift right B and subtract; enter 1 in Q
                  ----000110       Remainder < B; enter 0 in Q
                  -----00110       Final remainder
```

HardwareImplementation forSigned-MagnitudeData

When the division is implemented in a digital computer, it is convenient to change the process slightly.Instead of shifting the divisorto the right, the dividend, or partial remainder, is shiftedto the left, thusleavingthetwonumbersintherequiredrelativeposition.SubtractionmaybeachievedbyaddingAtothe2's complement of B. The informationabouttherelativemagnitudes is then available from the end-carry.The hardware for implementing the division operation is identical to that required formultiplication andconsists of thecomponents shown in Figure. Register EAQ is now shifted to the left with 0 inserted intoQn and the previous valueof E lost.

The numerical example is repeated in Figure to clarify the proposeddivision process.The divisor is stored in the B register andthedouble-length dividendisstoredinregisters A and Q. The dividend is shifted to the left and the divisor is subtracted by adding its 2'scomplementvalue.TheinformationabouttherelativemagnitudeisavailableinE.IfE=1,itsignifiest hatAis greaterthanorequaltoB.

Divisor $B = 10001$, $\bar{B} + 1 = 01111$

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\bar{B}+1$		<u>01111</u>		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\bar{B}+1$		<u>01111</u>		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\bar{B}+1$		<u>01111</u>		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		<u>10001</u>		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\bar{B}+1$		<u>01111</u>		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\bar{B}+1$		<u>01111</u>		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		<u>10001</u>		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

A quotient bit 1 is inserted into Qn and the partial remainder is shifted to the left to repeat the process. If E =0, it signifies that A < B so the quotient in Qn remains a 0 (inserted during the shift). The value of B is thenadded to restore the partial remainder in A toits previousvalue.The partial remainder is shifted to the leftand the process is repeated againuntilallfivequotient bits are formed. Note that while the partialremainder is shifted left,the quotient bitsare shifted also and after five shifts, the quotient is in Q and thefinalremainderisinA.

Before showing the algorithm in flowchart form, we have to consider the sign of the result andpossibleoverflowcondition.Thesignofthequotientisdeterminedfromthesignsofthedividendan dthedivisor.If the two signs are alike, the sign of the quotient is plus. If they are unalike, the sign is minus. The sign of the remainder is the same as the sign of the dividend.

DivideOverflow

The division operation may result in a quotient with an overflow. This is not a problem when workingwith paper and pencil but is critical when the operation is implemented with hardware. This is becausethe length of registers is finite and will not hold a number that exceeds the standard length. To see this,consider a system that has 5-bit registers. We use one register to hold the divisor and two registers tohold the dividend. From the example of Figure we note that the quotient will consist of six bits if thefive most significant bits of the dividend constitute a number greater than the divisor. The quotient is tobe stored in a standard 5-bit register, so the overflow bit will require one more flip-flop for storing thesixth bit. This divide- overflow condition must be avoided in normal computer operations because theentirequotientwillbetoolongfortransferintoamemoryunitthathaswordsofstandardlength,t hatis, the same as the length of registers. Provisions to ensure that this condition is detected must beincludedineitherthehardwareorthe software ofthe computer,or inacombinationofthetwo.

Whenthedividendistwiceaslongasthedivisor,theconditionforoverflowcanbestatedasfol lows:A divide-overflow condition occurs if the high-order half bits of the dividend constitute a numbergreater than or equal to the divisor. Another problem associated with division is the fact that a divisionby zero must be avoided. The divide-overflow condition takes care of this condition as well. Thisoccurs because any dividend will be greater than or equal to a divisor which is equal to zero. Overflowcondition is usually detected when a special flip-flop is set. We will call it a divide-overflow flip-flopandlabelitDVF.

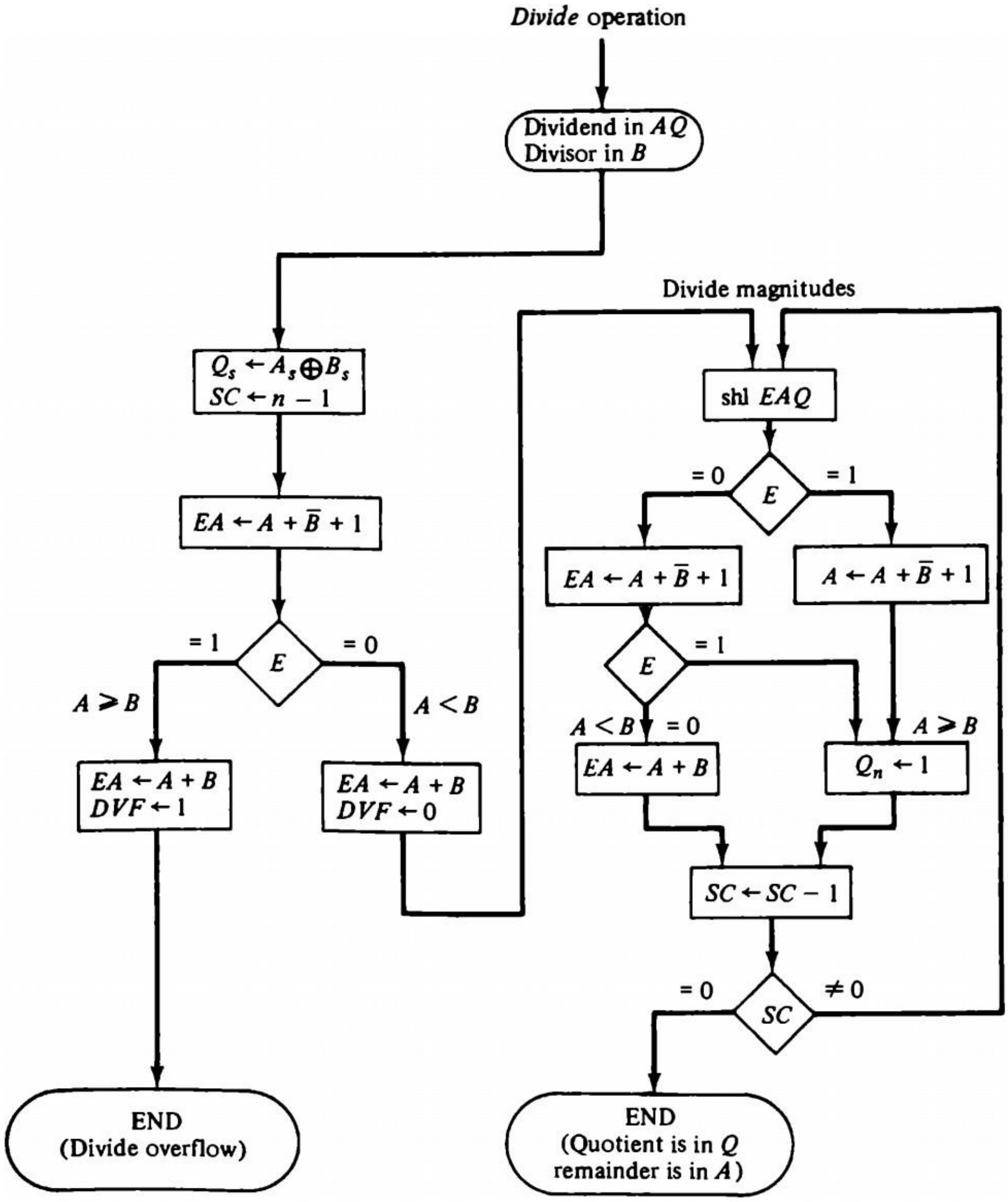
HardwareAlgorithm

The hardware divide algorithm is shown in the flowchart of Figure. The dividend is in A and Qand thedivisor in B. The sign of the result is transferred into Qs to be part of the quotient. A constant is set into thesequence counter SC to specify the number of bits in the quotient. As in multiplication, we assume thatoperands are transferred to registers fromamemory unit that has words of n bits. Since an operand mustbe stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n -1bits.

A divide-overflow condition is tested by subtracting the divisor in B from half of the bits of thedividend stored in A. If A is greater than or equal to B, the divide-overflow flip-flop DVF is set and theoperation is terminated prematurely. If A < B, no divide overflow occurs so the value of the dividend isrestored by adding B to A. The division of the magnitudes starts by shifting the dividend in AQ to theleft with the high-order bit shifted into E. If the bit shifted into E is 1, we know that EA > B becauseEA consists of a 1 followed by n-l bits while B consists of only n-1 bits. In this case, B must besubtracted from EA and 1 inserted into Qn for the quotient bit. Since register Aismissing the high-orderbit of the dividend (which is in E), its value is EA – 2^{n-1}. Adding to this value the 2's complement of Bresultsin(EA–2^{n-1})+(2^{n-1}-B)=EA-BThecarryfrom thisadditionisnottransferred toEifwewantEtoremaina1.

If the shift-left operation inserts a 0 into E, the divisor is subtracted by adding its2'scomplement valueand the carry is transferred into E. If E = 1, it signifies that A greater than or equal to B; therefore, Qn is setto 1. If E = 0, it signifies that A < B and the original number is restored by adding B to A. In the latter caseweleaveaoinQn(0wasinsertedduringtheshift).

This process is repeated again with register A holding the partial remainder. After n-l times, the quotientmagnitude is formed in register Q and the remainder is found in registerA.Thequotient sign is in Qs andthesign ofthe remainderinAs is the same astheoriginalsign ofthe dividend.



Divide operation
Dividend in AQ
Divisor in B
Divide magnitudes
Qs ← As ⊕ Bs
SC ← n − 1
EA ← A + B̄ + 1
E
= 1
= 0
A ≥ B
A < B
EA ← A + B
DVF ← 1
EA ← A + B
DVF ← 0
END
(Divide overflow)
shl EAQ
E
= 0
= 1
EA ← A + B̄ + 1
A ← A + B̄ + 1
E
= 1
A < B
= 0
A ≥ B
EA ← A + B
Qn ← 1
SC ← SC − 1
= 0
≠ 0
SC
END
(Quotient is in Q
remainder is in A)

3.9. Floating-PointArithmeticOperations:

Manyhigh-levelprogramminglanguageshaveafacilityforspecifyingfloating-pointnumbers.Anycomputer that has a compiler for such high-level programming language must have a provision for handlingfloating-point arithmetic operations. The operations are quite often included in the internal hardware. If nohardware is available for the operations, the compiler must be designed with a packageoffloating-pointsoftwaresubroutines.Althoughthe hardware method is more expensive, it is so much more efficient thanthe software method that floating-pointhardwareisincludedin4nost computersandisomittedonlyinverysmallones.

A floating-point number in computer registers consists of two parts: a mantissa m and an exponent e. Thetwopartsrepresentanumberobtainedfrommultiplyingmtimesaradixrraisedtothevalueofe;thus mx r^e.The mantissamay bea fraction oran integer. The location of the radix point and the value of theradixrareassumedandarenotincludedintheregisters.Forexample,assumeafractionrepresentationanda radix 10. The decimal number 537.25 is represented in a register with m = 53725 and e = 3 and isinterpreted to represent the floating- point number 0.53725 x 10^3. A floating-point number is normalized ifthemostsignificantdigitofthemantissaisnonzero.Inthiswaythemantissacontainsthemaximumpossible number of significant digits. A zero cannot be normalized because it does not have a nonzero digit.Itis represented in floating-pointbyall0's in the mantissa andexponent.

Arithmetic operations with floating-point numbers are more complicated than with fixed-point numbers andtheir execution takes longer and requires more complex hardware. Adding or subtracting two numbersrequires first an alignment of the radix point since the exponent parts must be made equal before adding orsubtracting the mantissas. The alignment is donebyshifting one mantissa while its exponent is adjusteduntilitis equaltothe otherexponent.Considerthesumofthe following floating-pointnumbers:

$$\begin{array}{r} .5372400 \times 10^{2} \\ +\ .1580000 \times 10^{-1} \end{array}$$

It is necessary that the two exponents be equal before the mantissas can be added. We can either shift thefirst number three positions to the left, or shift the second number three positions to the right. When themantissas are stored in registers, shifting to the left causes a loss of most significant digits. Shifting to theright causes a loss of least significant digits. The second methodis preferable because it only reduces theaccuracy, while the firstmethodmay cause anerror.The usual alignment procedure is to shift themantissa that has the smaller exponent to the right by a number of places equal to the difference between theexponents.Afterthis is done, the mantissascan beadded:

$$\begin{array}{r} .5372400 \times 10^2 \\ +.0001580 \times 10^2 \\ \hline .5373980 \times 10^2 \end{array}$$

When two normalized mantissas are added, the summaycontainanoverflowdigit.An overflow can becorrected easily by shifting the sum once to the right and incrementing the exponent. When two numbers aresubtracted,the resultmay containmostsignificantzerosas shownin thefollowingexample:

$$\begin{array}{r} .56780 \times 10^5 \\ -.56430 \times 10^5 \\ \hline .00350 \times 10^5 \end{array}$$

A floating-point number that has a 0 in the most significant position of the mantissa is said to have anunderflow. To normalize a number that containsan underflow,it is necessary to shiftthe mantissa to theleft and decrement the exponent until a nonzero digit appears in the first position. In the example above, it isnecessary to shift left twice to obtain .35000 x 10 3 . In most computers, a normalization procedure isperformedaftereachoperation to ensurethatallresults are in a normalizedform.

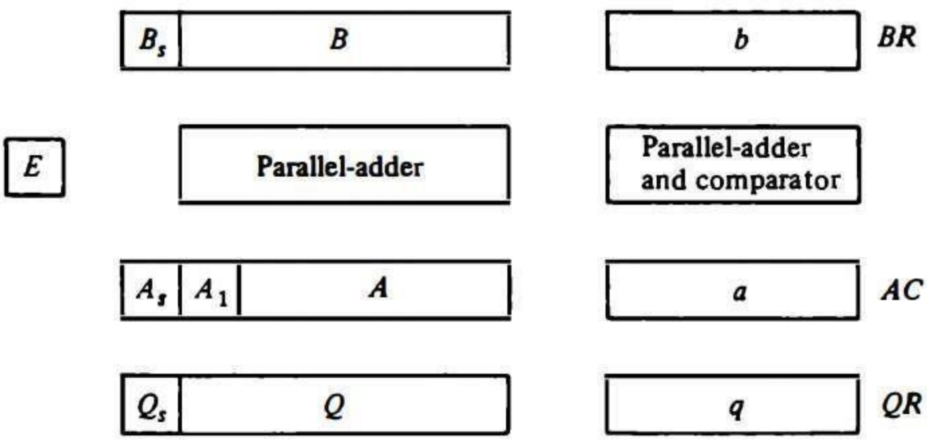
Floating-point multiplication and division do not require an alignment of the mantissas. The product can beformed by multiplying the two mantissas and adding the exponents. Division is accomplished by dividingthe mantissas and subtracting the exponents.Theoperations performed with the mantissas are the same asin fixed-point numbers, so the two can share the same registers and circuits. The operations performed withthe exponents are compare and increment (for aligning the mantissas), add and subtract (for multiplicationand division), and decrement (to normalize the result). The exponent may be represented in any one of thethreerepresentations:signed-magnitude, signed-2,s complement,orsigned-l'scomplement.

A fourth representation employed in many computers is known as a biased exponent. In this representation,the sign bit is removed from being a separate entity. The bias isapositivenumber that is added to eachexponent as the floating-point numberisformed,sothat internally all exponents are positive. Thefollowing example may clarify this type of representation.

Consider an exponent that ranges from -50 to 49.Internally, it is represented by two digits (without a sign) by adding to it a bias of 50. The exponentregistercontainsthe number e + 50, where e is the actual exponent. This way, the exponents are represented inregisters as positive numbers in the range of 00 to 99. Positive exponents in registers have the range ofnumbers from 99 to 50. The subtraction of 50 gives the positive values from 49 to 0. Negative exponents arerepresented in registers in the range from 49 to 00. The subtraction of 50 gives the negative values in therange of -1 to -50. The advantage ofbiased exponents is that they contain only positive numbers. It isthensimplertocomparetheirrelativemagnitudewithoutbeingconcernedwiththeirsigns.Asaconsequence,amagnitudecomparatorcanbeusedtocomparetheirrelativemagnitudeduringthealignment of the mantissa.Another advantage is that the smallest possible biasedexponent contains allzeros.Thefloating-pointrepresentationofzeroisthenazeromantissaandthesmallestpossibleexponent.

RegisterConfiguration

The register configuration for floating-point operations is quite similar to the layout for fixed- pointoperations. As a general rule, the same registers and adder used for fixed-point arithmetic are used forprocessing the mantissas. The difference lies in the way the exponents are handled.Theregisterorganization for floating-point operations is shown in Figure. There are three registers, BR, AC, andQR.Eachregisterissubdividedintotwoparts.Themantissaparthasthesameuppercaselettersymbolsas in fixed-point representation. The exponent part uses the corresponding lower-case letter symbol. Itis assumed that each floating-point number has a mantissa in signed-magnitude representation and abiased exponent. Thus the AC has a mantissa whose sign is in A s and a magnitude that is in A. Theexponent is in the part of the register denoted by the lowercase letter symbol a. The diagram showsexplicitly the most significant bit of A, labeled by A1. The bit in this position must be a 1 for thenumbertobenormalized.NotethatthesymbolACrepresentstheentireregister,thatis,theconcatenation of As, A1, and a. Similarly, register BR is subdivided into Bs, B, and b, and QR into Qs,Q,andq.



A parallel-adder adds the two mantissas and transfers the sum into A and the carry into E. A separateparallel-adder is used for the exponents. Since the exponents are biased, they do nothave a distinct sign bitbut are represented as a biased positive quantity. It is assumed that the floating-point numbers are so largethat the chance of an exponent overflow is very remote, and for this reason the exponent overflow will beneglected.Theexponentsarealsoconnectedtoamagnitudecomparatorthatprovidesthreebinaryo utputsto indicate their relative magnitude. The number in the mantissa willbe taken as a fraction, so the binarypoint isassumed to resideto the left of the magnitude part. Integer representation for floating-point causescertain scaling problems during multiplication and division. To avoid these problems, we adopt a fractionrepresentation. The numbers in the registers are assumed to be initially normalized. After each arithmeticoperation, the result will be normalized. Thus allfloating-pointoperandscoming from and going to thememoryunitare alwaysnormalized.

AdditionandSubtraction:

Duringadditionorsubtraction,thetwofloating-pointoperandsareinACandBR.Thesumordifferenceisformedin theAC.Thealgorithmcan be divided into fourconsecutive parts:

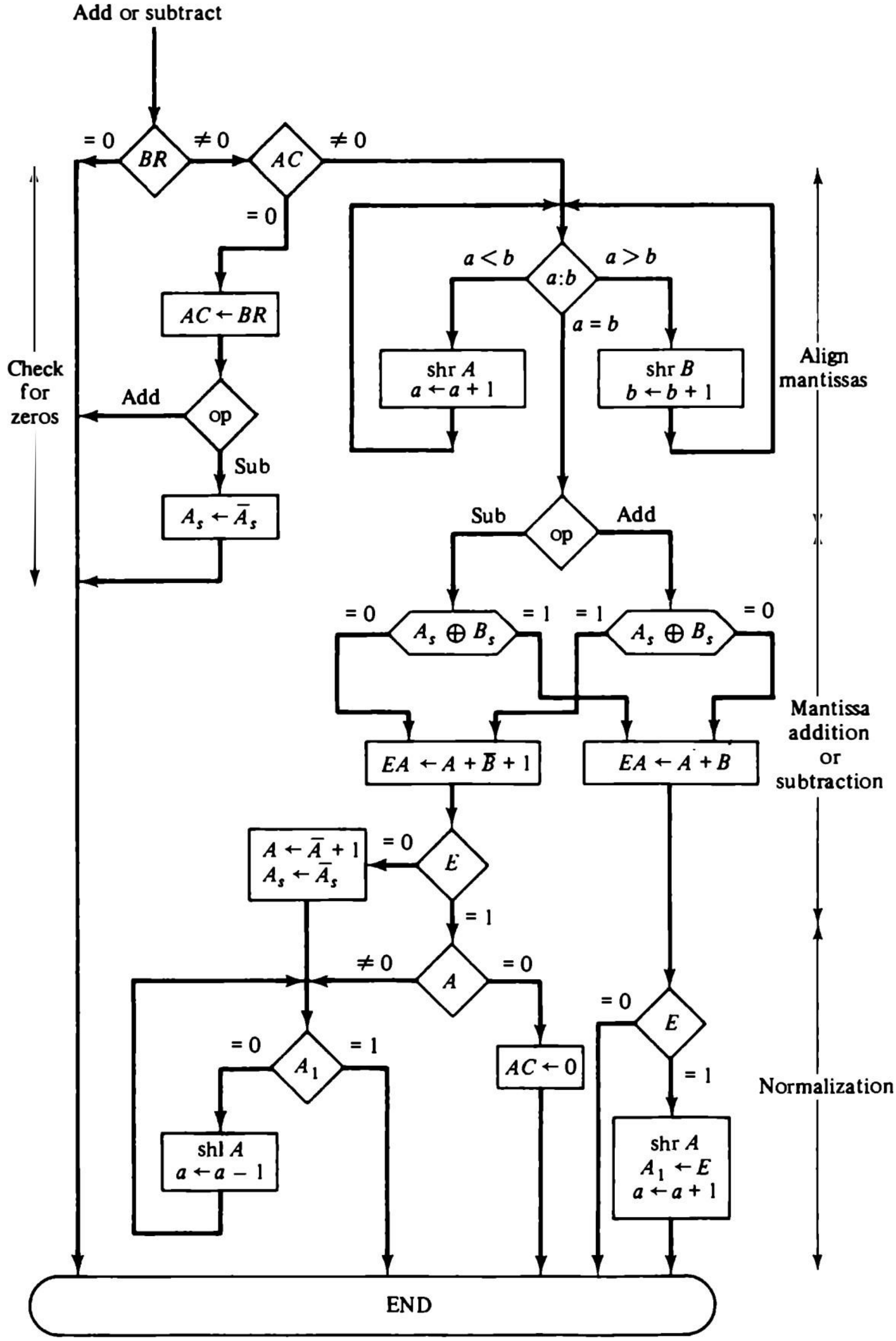
- Checkforzeros.
- Alignthemantissas.
- Addorsubtractthemantissas.
- Normalizetheresult.

Afloating-pointnumberthatiszerocannotbenormalized.Ifthisnumberisusedduringthecomputation,the result may also be zero. Instead of checkingforzerosduringthenormalization process we check forzeros at the beginning and terminate theprocessifnecessary. The alignment of the mantissas must becarriedoutpriortotheiroperation.Afterthemantissasareaddedorsubtracted,theresultmaybeunn ormalized.Thenormalization procedure ensures that the result is normalized prior to its transfer tomemory.

The flowchart for addingorsubtracting two floating-point binary numbers is shown in Figure.If BR isequal to zero, theoperation isterminated, with the value in the AC being the result. If AC is equal tozero,wetransferthecontentofBRintoACandalsocomplementitssignifthenumbersaretobesubtracted.Ifneithernumberis equalto zero,weproceedto align the mantissas.

The magnitude comparator attached to exponents a and b provides three outputs that indicate their relativemagnitude. If the two exponents are equal, we go to perform the arithmetic operation. If the exponents arenotequal,themantissahavingthesmallerexponentisshiftedtotherightanditsexponentincremented.Thisprocess is repeated untilthe two exponents are equal.

The addition and subtraction of the two mantissas is identical to the fixed-point addition and subtractionalgorithm. The magnitude part is added or subtracted depending on the operationand the signs of the twomantissas. If an overflow occurs when the magnitudes are added, it is transferred into flip-flop E. If E isequal to 1, the bit is transferred into A1 and all other bits of A are shifted right. The exponent must beincremented to maintain the correct number. No underflow may occur in this case because the originalmantissa that was not shifted during the alignment was already in a normalized position. If the magnitudeswere subtracted, the resultmay be zero or may have an underflow. If the mantissa is zero,the entirefloating-point numberin the AC is made zero. Otherwise, the mantissa must have at least one bit that isequal to 1. The mantissa has an underflow if the most significant bit in position A1 is 0. In that case,themantissa is shifted left and the exponent decremented. The bit in A1 is checked again and the process isrepeateduntilitisequalto 1.WhenA1=1,the mantissa isnormalized and theoperationiscompleted.


Add or subtract
BR
= 0
≠ 0
AC
= 0
≠ 0
AC ← BR
op
Add
Sub
$A_s \leftarrow \overline{A}_s$
Check for zeros
a:b
a < b
a > b
a = b
shr A
a ← a + 1
shr B
b ← b + 1
Align mantissas
op
Sub
Add
$A_s \oplus B_s$
= 0
= 1
= 1
$A_s \oplus B_s$
= 0
$EA \leftarrow A + \overline{B} + 1$
$EA \leftarrow A + B$
Mantissa addition or subtraction
E
= 0
$A \leftarrow \overline{A} + 1$
$A_s \leftarrow \overline{A}_s$
= 1
A
≠ 0
= 0
A_1
= 0
= 1
shl A
a ← a − 1
AC ← 0
E
= 0
= 1
shr A
$A_1 \leftarrow E$
a ← a + 1
Normalization
END

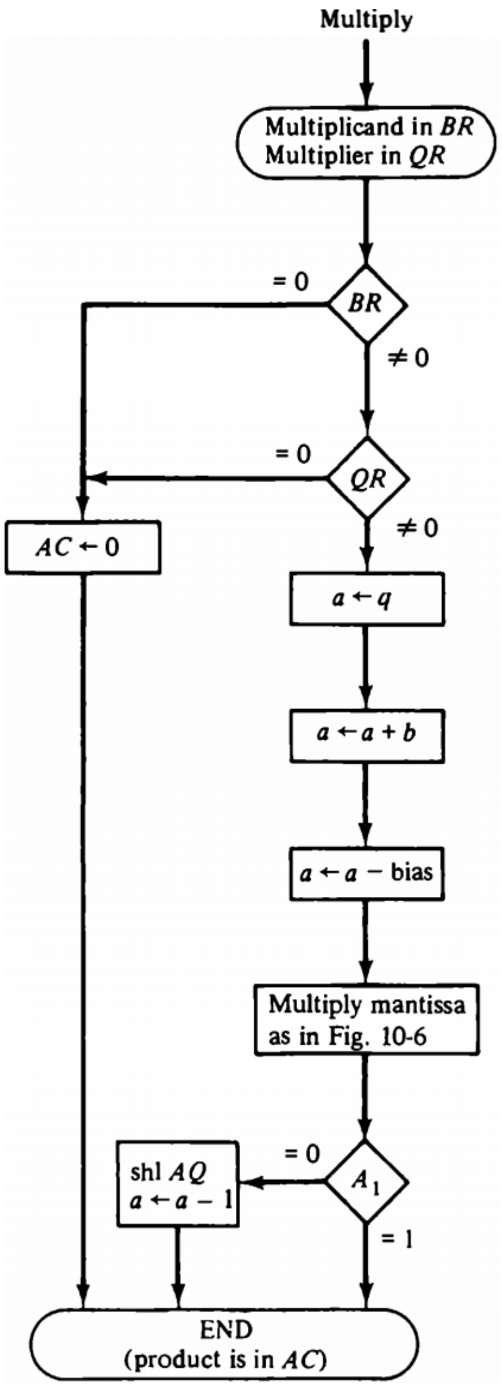
Multiplication

The multiplication of two floating-point numbers requires that we multiplythe mantissas andadd theexponents. No comparison of exponents or alignment of mantissas is necessary. The multiplication of themantissas is performed in the same way as in fixed-point to provide a double-precision product. The double-precision answer is used in fixed-point numberstoincrease the accuracy of the product. In floating-point,the range of a single-precision mantissa combined with the exponent is usually accurate enough so that onlysingle-precisionnumbersare maintained. Thusthe half most significant bits of the mantissa product andtheexponentwillbe taken togetherto forma single-precisionfloating-pointproduct.

Themultiplicationalgorithmcanbesubdividedintofourparts:

- Checkforzeros.
- Addtheexponents.
- Multiplythemantissas.
- Normalizetheproduct.

Steps 2 and 3 can be done simultaneously if separate adders are available for the mantissas and exponents.The flowchart for floating-point multiplication isshowninFigure.Thetwo operands are checked todetermine if they contain a zero. If either operand is equal to zero, the product in the AC is set to zero andthe operation is terminated. Ifneitherof the operands is equal to zero, the process continues with theexponentaddition. Theexponentof the multiplier is in q and the adder is between exponents a and b. It isnecessary totransfertheexponentsfrom q to a, add the two exponents, and transfer the sum into a. Sinceboth exponents are biased by the addition of a constant, the exponent sum will have double this bias. Thecorrect biased exponent for the product is obtained by subtracting the bias number from the sum. Themultiplication of the mantissas is done as in the fixed-point case with the product residing in A and Q.Overflowcannotoccurduringmultiplication,sothereisnoneedtocheckforit.



The product may have an underflow, so the most significant bit in A is checked. If it is a 1, the

productisalreadynormalized.Ifitisa0,themantissainAQisshiftedleftandtheexponentdecremented.Note that only one normalization shift is necessary. The multiplier and multiplicand were originallynormalized and contained fractions. The smallest normalized operand is 0.1, so the smallest possibleproduct is 0.01. Therefore, only one leading zero may occur. Although the low-order half of themantissa is in Q, we do not use it for the floating-point product. Only the value in the AC is taken astheproduct.

Division

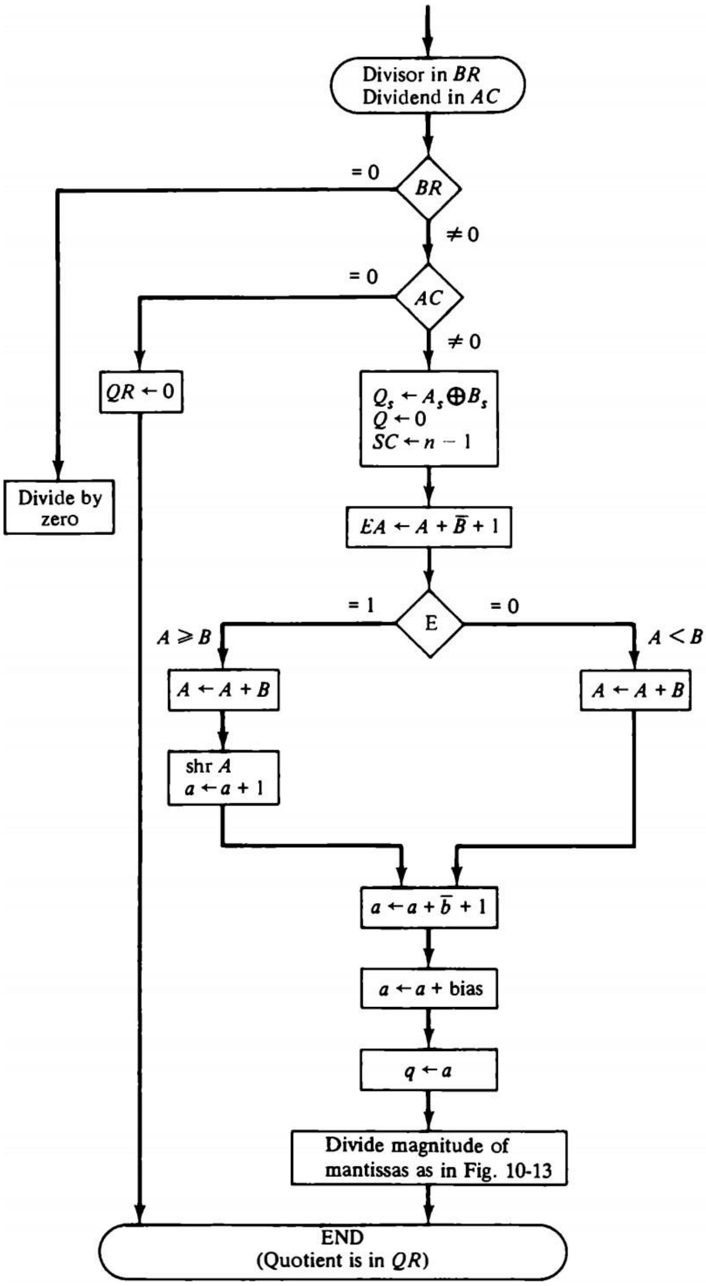
Floating-point division requires that the exponents be subtractedandthemantissasdivided. The mantissadivisionisdoneasinfixed-pointexceptthatthedividendhasasingle-precisionmantissathatisplacedinthe AC. Remember that the mantissa dividend is a fraction and not an integer. For integer representation, asingle-precision dividend must be placed in register Q and register A must be cleared. The zeros in A are totheleftofthebinarypointandhavenosignificance.Infractionrepresentation,asingle-precisiondividendis placed in register A and register Q is cleared. The zeros in Q are to the right of the binary point and haveno significance. The check for divide-overflow is the same as in fixed-point representation. However, withfloating-point numbers the divide-overflow imposes no problems. Ifthedividendisgreaterthan or equalto the divisor, the dividend fraction is shifted to the right and its exponent incremented by 1. For normalizedoperands this is a sufficient operation to ensure that no mantissa divide-overflow will occur. The operationabove isreferredtoasadividendalignment. The division of two normalized floating-point numbers willalways result in a normalized quotient provided that a dividend alignment is carried out before the division.Therefore,unliketheotheroperations,thequotientobtainedafterthedivisiondoesnotrequireanormalization.

Thedivisionalgorithmcanbesubdividedintofiveparts:

- Checkforzeros.
- Initializeregistersandevaluatethesign.
- Alignthedividend.
- Subtracttheexponents.
- Dividethe mantissas.

The flowchart for floating-point division is shown in Figure. The two operands are checked for zero. If thedivisor is zero, it indicates an attempt to divide byzero,whichisanillegaloperation. The operation isterminated with an errormessage. An alternativeprocedure wouldbe to set the quotient in QR to the mostpositive number possible (if the dividendis positive) orto the most negative possible (if the dividend isnegative).IfthedividendinACiszero,thequotientinQRismadezero andtheoperationterminates.

Iftheoperandsarenotzero,weproceedtodeterminethesignofthequotientandstoreitinQs. The signof the dividend in As is left unchanged to be the sign of the remainder. The Q register is cleared and thesequence counter SC is set to a number equal to the number of bits in the quotient. The dividendalignment is similar to the divide-overflow check in the fixed-point operation. The proper alignmentrequires that the fraction dividend be smaller than the divisor. The two fractions are compared by asubtraction test. The carry in E determines their relative magnitude. The dividend fraction is restored toits original value by adding the divisor. If A is greater than or equal to B, it is necessary to shift A onceto the right and increment the dividend exponent. Since both operands are normalized, this alignmentensures that A < B. Next, the divisor exponent is subtracted from the dividend exponent. Since bothexponents were originally biased, the subtraction operation gives the difference without the bias. Thebiasisthen addedand the resulttransferred intoqbecause thequotientisformedin QR.Themagnitudes of the mantissas are divided as in the fixed-point case. After the operation, the mantissaquotient resides in Q and the remainder in A. The floating-point quotient is already normalized andresidesinQR.



The exponent of the remainder should be the same as the exponent of the dividend. The binary point for theremainder mantissa lies (n—1) positions to the left of A1. The remainder can be converted to a normalizedfractionbysubtractingn-1fromthedividendexponentandbyshiftanddecrementuntilthebitinA1isequalto1.Thisis notshownin the flowchart.

3.10. DecimalArithmeticOperations:

Thealgorithmsforarithmeticoperationswithdecimaldataaresimilartothealgorithmsforthec orresponding operations with binary data. In fact, except for a slight modification in the multiplication anddivision algorithms, the same flowcharts can be used for both types of data provided that we interpret themicro-operation symbols properly. Decimal numbers in BCD are stored in computer registers in groups offour bits. Each 4-bit group represents a decimal digitand must be taken as a unit when performing decimalmicro-operations.Forconvenience, wewill use the same symbols for binary and decimal arithmeticmicro-operations but give them a different interpretation. As shown in Table, a bar over the register lettersymbol denotes the 9's complement of the decimal number stored in the register. Adding 1 to the 9'scomplementproducesthe 10'scomplement.

Thus, for decimal numbers, the symbol A <- A+B'+1 denotes a transfer of thedecimal sum formed byadding the original content A to the 10's complement of B. The use ofidenticalsymbols for the 9'scomplementandthel'scomplementmay beconfusingifboth typesofdataareemployedin thesamesystem

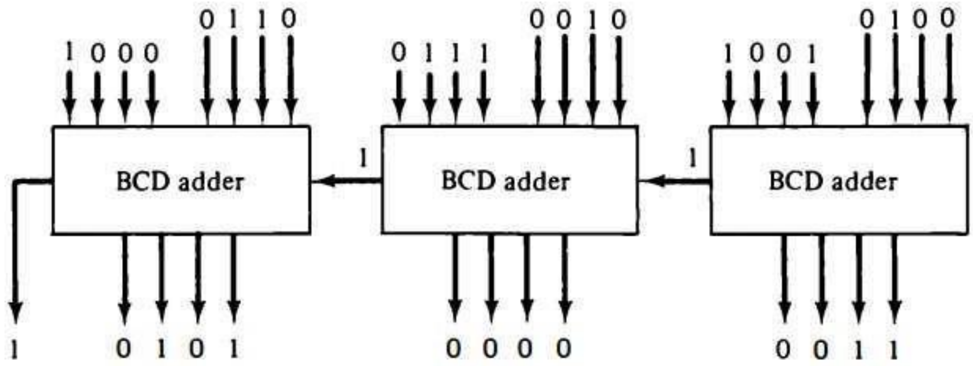
Incrementing or decrementing a register is the same for binary and decimal except for thenumber of statesthat the register is allowed to have. A binary counter goes through 16 states,from 0000 to 1111, whenincremented. A decimal counter goes through 10 states from 0000 to 1001 and back to 0000, since 9 is the lastcount. Similarly, a binary counter sequences from 1111to 0000 when decremented. A decimal counter goesfrom 1001 to 0000. A decimal shift right or left is preceded by the letter *d* to indicate a shift over the fourbitsthathold the decimal digits.As a numerical illustration consider a register *A* holding decimal 7860 in BCD.The bit pattern of the 12 flip-flops is 0111 1000 0110 0000. The micro-operation *dshr A* shifts the decimalnumber one digit to the right to give 0786. This shift is over the four bits and changes

Symbolic Designation	Description
$A \leftarrow A + B$	Add decimal numbers and transfer sum into *A*
$\overline{B}$	9's complement of *B*
$A \leftarrow A + \overline{B} + 1$	Content of *A* plus 10's complement of *B* into *A*
$Q_L \leftarrow Q_L + 1$	Increment BCD number in Q_L
dshr *A*	Decimal shift-right register *A*
dshl *A*	Decimal shift-left register *A*

the content of theregisterinto 0000 0111 10000110.

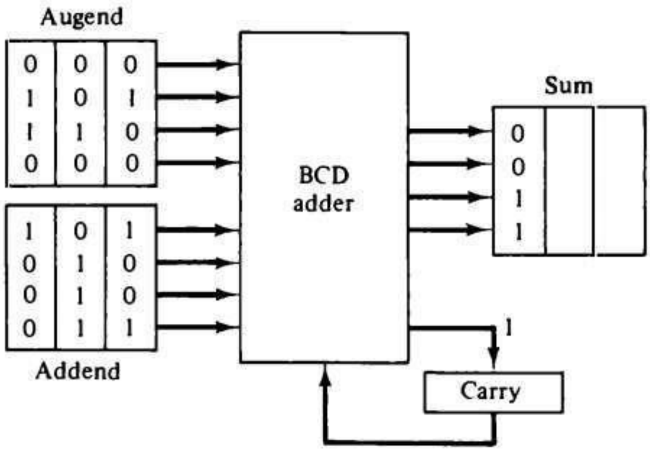
AdditionandSubtraction

The algorithm for addition and subtraction of binary signed-magnitude numbers applies also to decimalsigned-magnitudenumbersprovidedthat we interpret themicro-operationsymbols inthe propermanner. Similarly, the algorithm for binary signed-2's complement numbers applies to decimal signed-10's complement numbers. The binary data must employ a binary adder and a complementer. Thedecimal data must employ a decimal arithmetic unit capable of adding two BCD numbers and formingthe 9's complement of the subtrahend. Decimal data can be added in three different ways, as shown inFigure. The parallel method uses a decimal arithmetic unit composed of as many BCD adders as therearedigitsinthenumber.Thesumisformedin parallelandrequiresonly onemicro-operation.

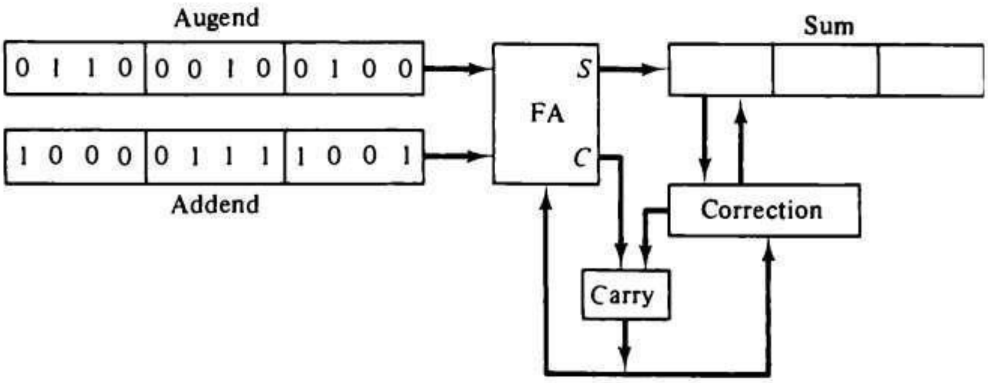


(a) Parallel decimal addition: 624 + 879 = 1503

In the digit-serial bit-parallel method, the digits are applied toasingleBCDadderserially, while the bitsof each coded digit are transferred in parallel. The sum is formed by shifting the decimal numbers throughthe BCD adder one at a time. For k decimal digits, this configuration requires k micro-operations, one foreach decimal shift. In the all serial adder,thebitsare shifted one at a time through a full-adder. The binarysum formed after four shifts must be corrected into a valid BCD digit. If the binary sum is greater than orequalto 1010, the binarysum is corrected by adding to it 0110 and generating a carry for the next pair ofdigits.



(b) Digit-serial, bit-parallel decimal addition

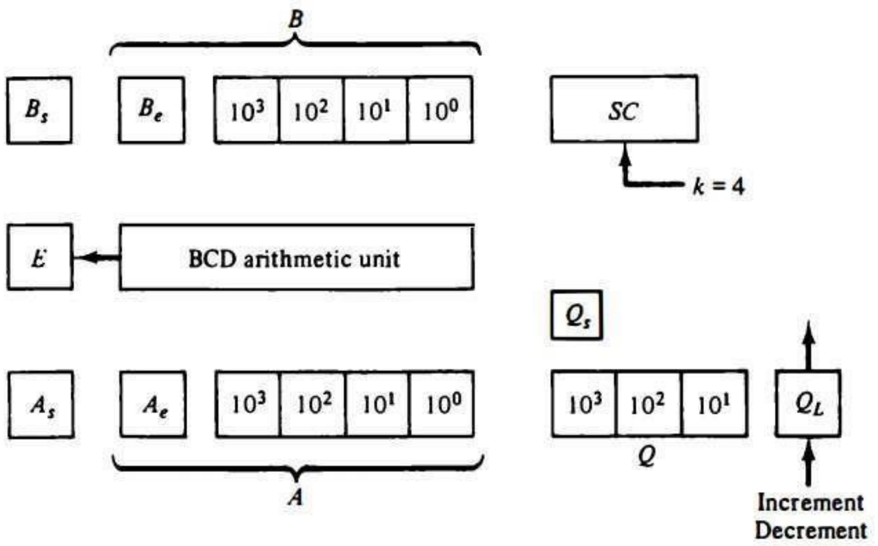


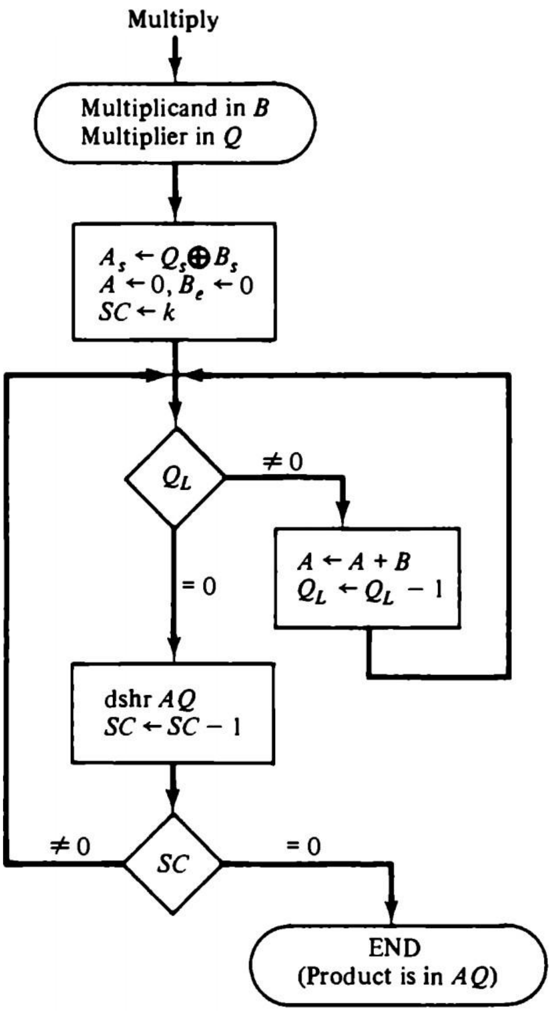
(c) All serial decimal addition

The parallel method is fast but requires a large number of adders. The digit-serial bit-parallel methodrequires only one BCD adder, which is shared by all the digits. It is slower than the parallel method becauseofthetimerequiredtoshiftthedigits.Theallserialmethodrequiresaminimumamountofequipmentbutisveryslow.

Multiplication

Themultiplicationoffixed-pointdecimalnumbersis similartobinaryexcept fortheway thepartialproducts are formed. A decimal multiplier has digits that range in value from 0 to 9, whereas a binarymultiplier has only 0 and 1 digits. In thebinarycase,themultiplicand is added to the partial product ifthe multiplier bit is1. In the decimal case, the multiplicandmust be multiplied by the digit multiplier andthe result added to the partial product. This operationcan be accomplishedby adding the multiplicand tothe partial product anumber of times equalto the value of the multiplier digit. The registers organizationfor the decimal multiplication is shown in Figure. We are assuming here four-digit numbers, with each digitoccupying four bits, for a total of 16 bits for each number. There are three registers, A, B, and Q, eachhavinga corresponding signflip-flop As, Bs, andQs.





Registers A and B have four more bits designated by *Ae* and *Be* that provide an extension of one more digitto the registers. The BCD arithmetic unitadds the five digits in parallelandplacesthe sum in the five-digit A register.The end-carrygoesto flip-flop E. The purpose of digit Ae is to accommodate anoverflow while adding the multiplicand to the partial product during multiplication. The purpose of digit *Be*is to form the 9's complement of the divisor when subtracted from the partial remainder during the divisionoperation.The leastsignificant digitin register Q is denoted by QL. This digit can be incremented ordecremented.

Adecimaloperandcomingfrommemoryconsistsof17bits.Onebit(thesign)istransferred to Bs and the magnitude of the operand is placed in the lower 16 bits of B. Both *Be* and *Ae* arecleared initially. The result of the operation is also 17 bits long and does not use the*Ae* part of the Aregister. The decimal multiplication algorithm is shown in Figure. Initially, the entire A register and *Be* arecleared and the sequence counter SC is set to a number *k* equal to the number of digits in the multiplier. Thelow-order digit of the multiplier in QL is checked. If it is not equal to 0, the multiplicand in B is added to thepartial product in A once and QL isdecremented.QLischecked again and the process is repeated until itis equal to 0. In this way, the multiplicand in B is added to the partial product a number of times equal tothemultiplierdigit.Anytemporaryoverflowdigitwillresidein*Ae*andcanrangeinvaluefrom0to9.Next, the partial product and the multiplier are shifted once to the right. This places zero in *Ae* and transfersthenext multiplier quotient into QL. The process is then repeated *k* times to form a double-lengthproductinAQ.

Division

Decimaldivisionissimilartobinarydivisionexceptofcoursethatthequotientdigitsmayhaveanyofthe10 values from 0 to 9. In the restoring division method, the divisor is subtracted from the dividend or partialremainder as many times asnecessaryuntilanegativeremainder results. The correct remainder is thenrestored by adding the divisor. The digit in the quotient reflects the number of subtractions up to butexcluding the one that caused the negative difference. The decimal division algorithm is shown in Figure. Itis similar to the algorithm with binary data except forthe way the quotientbits are formed. The dividend(orpartial remainder) is shifted to the left, with its most significant digitplacedin *Ae*.The divisoris thensubtractedby adding its 10's complement value. Since *Be* is initially cleared, its complement value is 9 asrequired. The carry in E determines the relative magnitude of A and B.If E =0,itsignifies that A < B. Inthis case the divisor is added to restore the partial remainder and Q l stays at0(inserted there during theshift). If E = 1, it signifies that A greater than or equal to B.

Thequotient digit in QL is incremented onceandthedivisor subtractedagain.Thisprocessisrepeateduntilthesubtractionresultsinanegativedifference which is recognized by E being 0. When this occurs, the quotient digit is not incremented but thedivisor is added to restore the positive remainder. In this way, the quotient digit is made equal to the numberof times that the partial remainder "goes" into the divisor. The partial remainder and the quotient bits areshifted once to the left and the process is repeated k times to form k quotient digits. The remainder isthenfoundinregisterAandthequotientisinregisterQ.ThevalueofEisneglected.

UNIT – 04 – Part - A

INPUT-OUTPUTORGANIZATION

The Input / output organization of computer depends upon the size of computer and theperipheralsconnectedtoit.The I/OSubsystemofthecomputer,provides anefficientmodeofcommunication betweenthecentral system and the outsideenvironment

Themostcommoninput outputdevicesare:

- ❖ Monitor
- ❖ Keyboard
- ❖ Mouse
- ❖ Printer
- ❖ Magnetictapes

The devices that are under the direct control of the computer are said to be connectedonline.

4.1. Input-Output Interface:

InputOutput Interfaceprovidesamethodfortransferringinformationbetweeninternalstorageandexternal I/Odevices.

Peripherals connected to a computer need special communication links for interfacing themwiththecentral processingunit.

The purpose of communication link is to resolve the differences that exist between thecentralcomputer and each peripheral.

TheMajorDifferencesare:-

- Peripherals are electro-mechanicaland electromagnetic devices and CPU andmemory are electronic devices. Therefore, a conversion of signal values may beneeded.
- The data transfer rate of peripherals is usually slower than the transfer rate of CPUandconsequently, asynchronization mechanism maybeneeded.
- Data codes and formats in the peripherals differ from the word format in the CPU andmemory.
- The operating modes of peripherals are different from each other and must becontrolled so as not to disturb the operation of other peripherals connected to theCPU.

To Resolve these differences, computer systems include special hardware componentsbetweentheCPUandPeripheralsto supervisesandsynchronizesallinputandouttransfers

These components are called Interface Units because they interface between theprocessorbusand the peripheral devices.

I/OBUSand InterfaceModule:

Itdefinesthetypicallinkbetweentheprocessorandseveralperipherals.

The I/O Bus consists of data lines, address lines and control lines.TheI/Obusfromtheprocessorisattachedtoallperipheralsinterface.

To communicate with a particular device, the processor places a device address on addresslines.

Each Interface decodes the address and control received from the I/O bus, interprets them forperipheralsand providessignals fortheperipheralcontroller.

Itisalsosynchronizesthedataflowandsupervisesthetransfer betweenperipheralandprocessor.

Eachperipheralhasitsowncontroller.

For example, the printer controllercontrols the paper motion, the print timingThecontrollinesarereferredasI/Ocommand.Thecommandsareasfollowing:

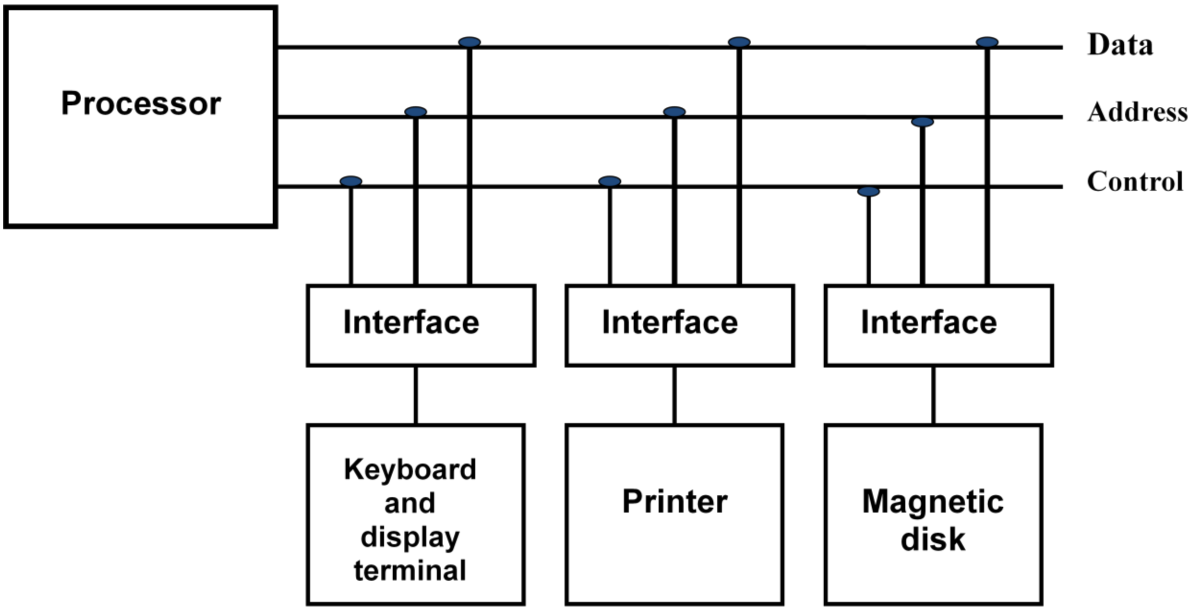
Control command- A control command is issued to activate the peripheral and to inform itwhatto do.

Status command- A status command is used to test various status conditions in the interfaceandthe peripheral.

Data Output command- A data output command causes the interface to respond bytransferringdata from the bus into oneofits registers.

Data Input command-Thedata input commandis the oppositeof thedataoutput.

In this case the interface receives on item of data from the peripheral and places it in itsbuffer register.I/O Versus MemoryBus



Connection of I/O bus to input-output devices

To communicate with I/O, the processor must communicate with the memory unit. Like theI/O bus, the memory bus contains data, address and read/write control lines. There are 3 waysthatcomputer buses canbeusedto communicatewithmemoryandI/O:

- Usetwo Separatebuses , oneformemoryand otherfor I/O.

- Useonecommon busforbothmemoryandI/Obut separate controllines foreach.
- Use one common bus for memory and I/O with common control lines.I/OProcessor

In the first method, the computer has independent sets of data, address and control busesone for accessing memory and other for I/O. This is done in computers that provides aseparate I/O processor (IOP). The purpose of IOP is to provide an independent pathway forthetransfer ofinformation betweenexternal deviceand internal memory.

4.2. AsynchronousDataTransfer:

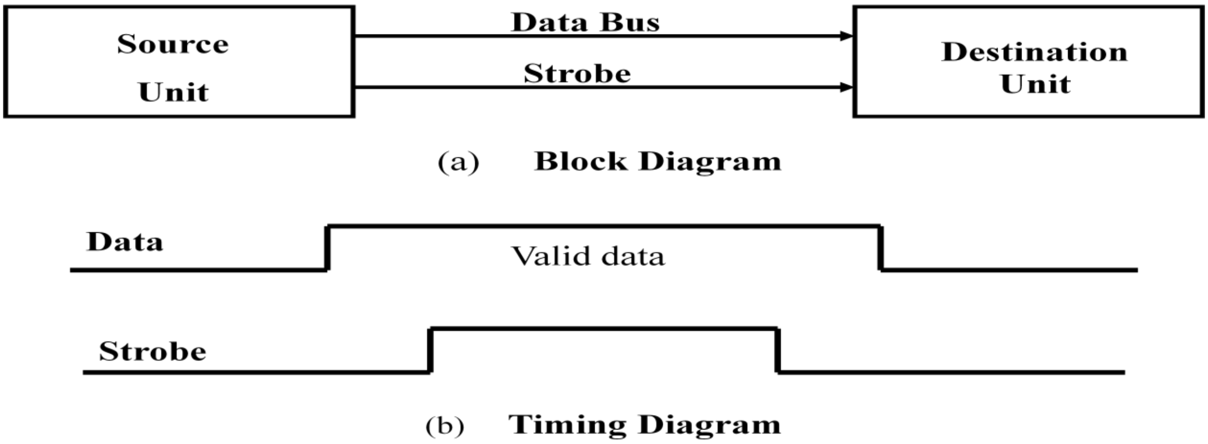
This Scheme is used when speed of I/O devices do not match with microprocessor, andtiming characteristics of I/O devices is not predictable. In this method, process initiates thedevice and check its status. As a result, CPU has to wait till I/O device is ready to transferdata.WhendeviceisreadyCPUissuesinstructionfor I/Otransfer.Inthismethodtwotypesoftechniques areusedbased on signals beforedatatransfer.

- ✓ StrobeControl
- ✓ Handshaking

StrobeSignal :

The strobe control method of Asynchronous data transfer employs a single control line totime eachtransfer. Thestrobemaybeactivated byeither thesourceor thedestination unit.

DataTransferInitiatedbySourceUnit:



Source-Initiated strobe for Data Transfer

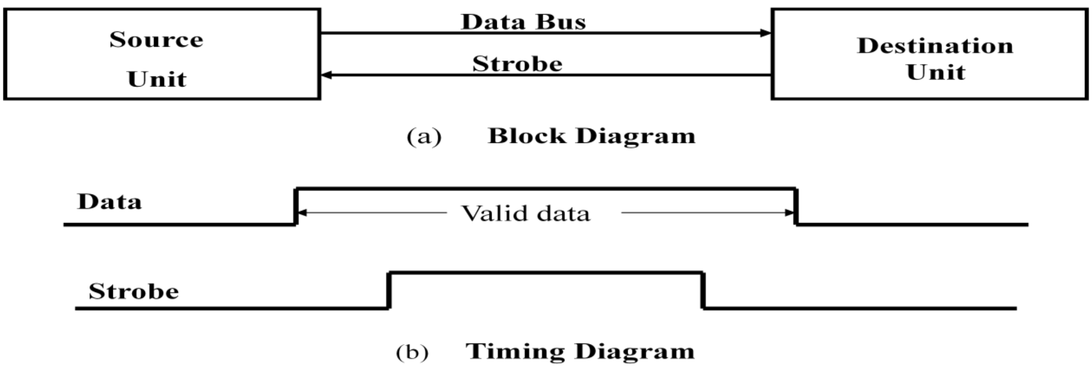
In the block diagram fig. (a), the data bus carries the binary information from source todestination unit. Typically, the bus has multiple lines to transfer an entire byte or word. Thestrobeis asingle line thatinforms thedestination unit whenavaliddata word isavailable.

The timing diagram fig. (b) the source unit first places the data on the databus. The information on the data bus and strobe signal remain in the active state to allow thedestinationunit to receive thedata.

DataTransferInitiatedbyDestinationUnit:

In this method, the destination unit activates the strobe pulse, to informing the source toprovide the data. The source will respond by placing the requested binary information on thedata bus.

The data must be valid and remain in the bus long enough for the destinationunit to accept it. When accepted the destination unit then disables the strobe and the sourceunitremoves thedata from thebus.



Destination-Initiated strobe for Data Transfer

DisadvantageofStrobeSignal

The disadvantage of the strobe method is that, the source unit initiates the transfer has no wayof knowing whether the destination unit has actually received the data item that was places inthe bus. Similarly, a destination unit that initiates the transfer has no way of knowing whetherthe source unit has actually placed the data on bus. The Handshaking method solves thisproblem.

Handshaking:

The handshaking method solves the problem of strobe method by introducing a secondcontrolsignal that provides a replyto theunit thatinitiatesthetransfer.

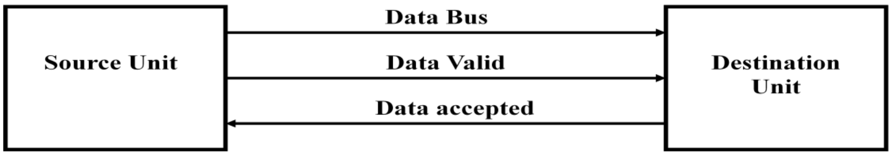
PrincipleofHandshaking:

Thebasicprinciple of the two-wirehandshakingmethod ofdata transfer isas follow:

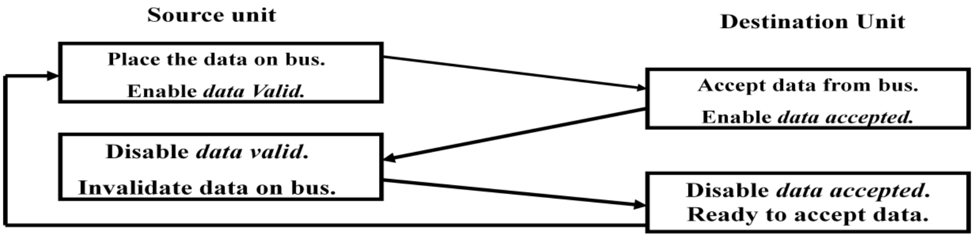
One control line is in the same direction as the data flows in the bus from the source todestination. It is used by source unit to inform the destination unit whether there a valid datainthebus. Theothercontrollineis intheother directionfrom thedestinationto thesource.Itis used by the destination unit to inform the source whether it can accept the data. Thesequenceof controlduringthe transferdepends ontheunit thatinitiates the transfer.

SourceInitiatedTransferusingHandshaking:

The sequence of events shows four possible states that the system can be at any given time.The source unit initiates the transfer by placing the data on the bus and enabling its *data valid*signal. The *data accepted* signal is activated by the destination unit after it accepts the datafrom the bus. The source unit then disables its *data accepted* signal and the system goes intoitsinitial state.



(a) Block Diagram

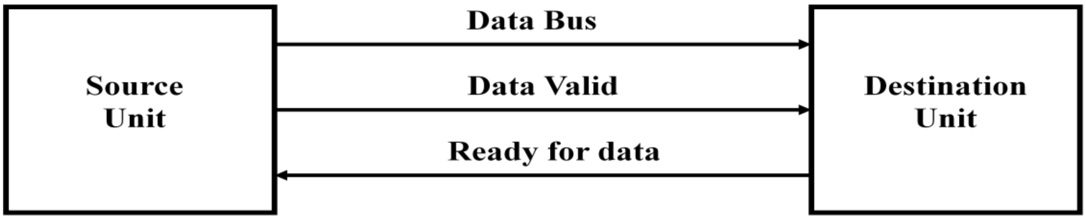


(b) Sequence of events

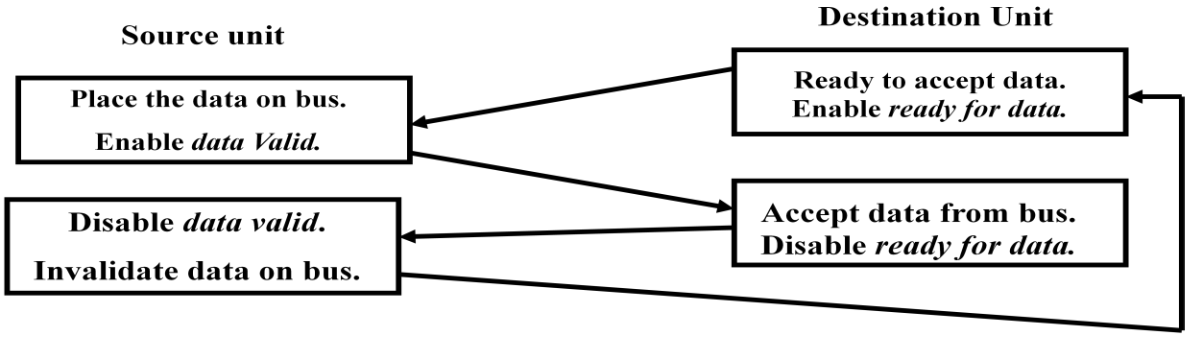
Destination InitiatedTransferUsingHandshaking:

The name of the signal generated by the destination unit has been changed to *ready for data*to reflects its new meaning. The source unit in this case does not place data on the bus untilafter it receives the *ready for data* signal from the destination unit. From there on, thehandshakingprocedure follows the samepattern as in thesourceinitiated case.

The only difference between the Source Initiated and the Destination Initiated transfer is intheirchoiceofInitial sate.



(a) Block Diagram



(b) Sequence of events

Destination-Initiated transfer using Handshaking

AdvantageoftheHandshakingmethod:

- The Handshaking scheme provides degree of flexibility and reliability because thesuccessfulcompletionof datatransfer reliesonactiveparticipation bybothunits.
- If any of one unit is faulty, the data transfer will not be completed. Such an error canbe detected by means of a *Timeout mechanism* which provides an alarm if the data isnotcompleted within time.

4.3. ModesofData Transfer:

Transfer of data is required between CPU and peripherals or memory or sometimes betweenany two devices or units of your computersystem. To transfer a data from one unit toanother one should be sure that both units have proper connection and at the time of datatransfer the receiving unit is not busy. This data transfer with the computer is InternalOperation.

All the internal operations in a digital system are synchronized by means of clock pulsessuppliedbyacommonclock pulseGenerator. Thedata transfer can be

- Synchronous

❖ Asynchronous

When both the transmitting and receiving units use same clock pulse then such a data transferiscalled Synchronousprocess. Ontheotherhand,ifthethereis notconcept ofclockpulsesand the sender operates at different moment than the receiver then such a data transfer iscalledAsynchronous datatransfer.

The data transfer can be handled by various modes. some of the modes use CPU as anintermediate path, others transfer the data directly to and from the memory unit and this canbehandled by3 followingways:

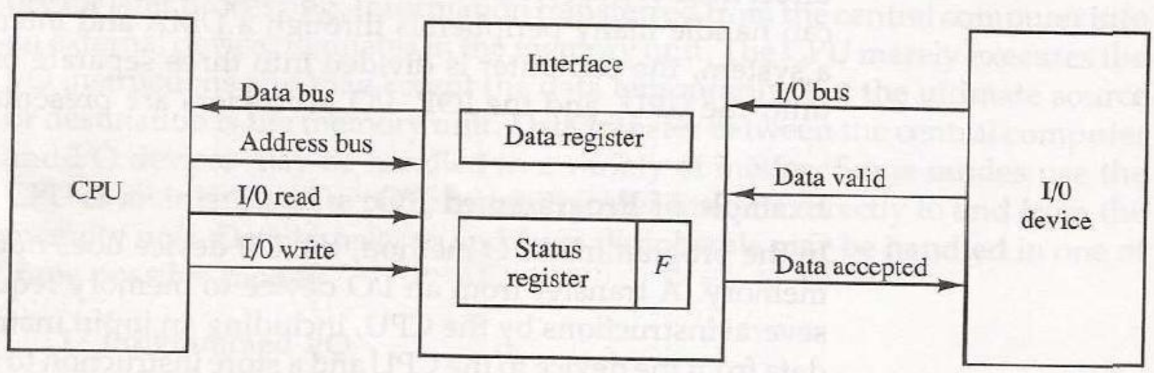
❖ ProgrammedI/O
❖ Interrupt-InitiatedI/O
❖ DirectMemoryAccess (DMA)

4.3.1. ProgrammedI/OMode:

InthismodeofdatatransfertheoperationsaretheresultsinI/Oinstructionswhichisapart of computer program. Each data transfer is initiated by a instruction in the program.Normallythe transfer isfrom aCPU registerto peripheral deviceorvice-versa.

Once the data is initiated the CPU starts monitoring the interface to see when next transfercan made. The instructions of the program keep close tabs on everything that takes place intheinterfaceunit and theI/O devices.

Figure Data transfer from I/O device to CPU.

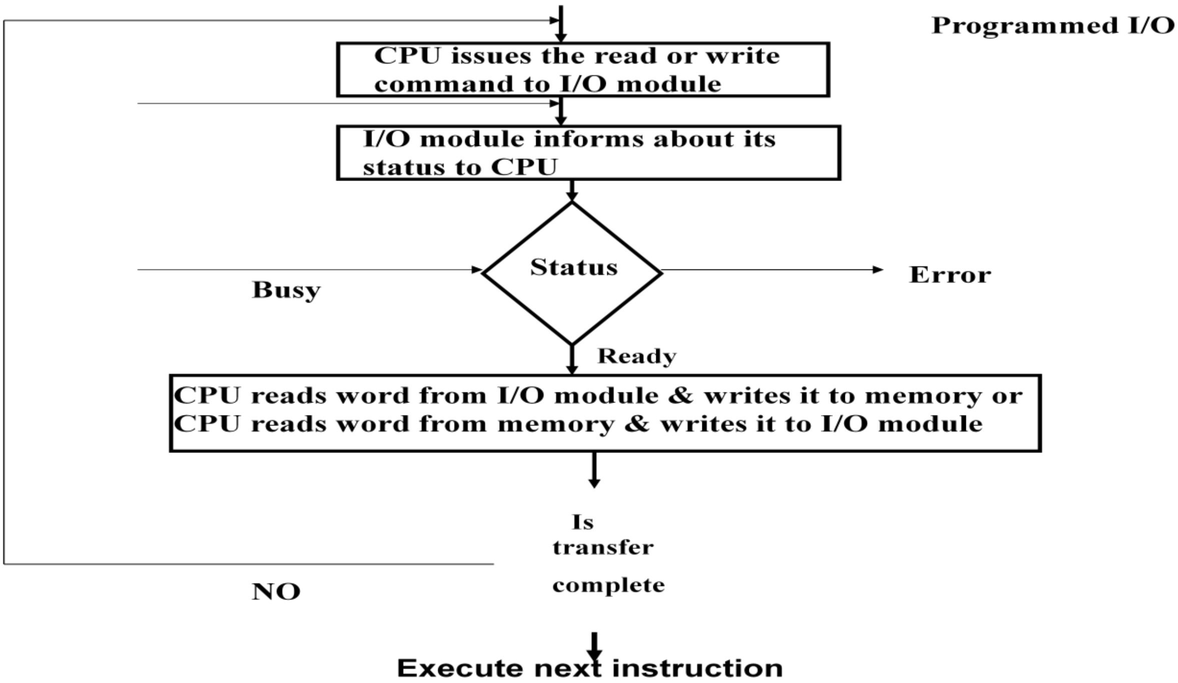


F = Flag bit

✓ Thetransferofdata requiresthreeinstructions:

1. Read the status register.
2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
3. Read the data register.

Inthistechnique CPUisresponsibleforexecutingdatafromthememoryforoutputandstoringdata in memoryforexecutingof ProgrammedI/O asshown inFlowchart-:



DrawbackoftheProgrammed I/O:

The main drawback of the Program Initiated I/O was that the CPU has to monitor the units allthe times when the program is executing. Thus the CPU stays in a program loop until the I/Ounit indicates that it is ready for data transfer. This is a time consuming process and the CPUtimeis wasted alot in keeping aneyeto theexecutingof program.

ToremovethisproblemanInterrupt facilityand specialcommandsareused.

4.3.2. Interrupt-InitiatedI/O:

In this method an interrupt facility an interrupt command is used to inform thedevice aboutthe start and end of transfer. In the meantime the CPU executes other program. When theinterface determines that the device is ready for data transfer it generates an Interrupt Requestandsends it to the computer.

WhentheCPUreceivessuchan signal,ittemporarilystops theexecutionoftheprogram andbranches to a service program to process the I/O transfer and after completing it returns backtotask, what it was originallyperforming.

- InthistypeofIO,computerdoesnotchecktheflag.Itcontinuetoperformitstask.
- Wheneveranydevicewants theattention,it sends theinterrupt signal tothe CPU.
- CPU then deviates from what it was doing, store the return address from PC andbranchto theaddress of thesubroutine.
- Therearetwoways of choosingthebranch address:
    - ✓ VectoredInterrupt
    - ✓ Non-vectoredInterrupt

- InvectoredinterruptthesourcethatinterrupttheCPUprovidesthebranchinformation.T his

information iscalled interrupt vectored.

- In non-vectored interrupt, the branch address is assigned to the fixed address in thememory.

PriorityInterrupt:

- TherearenumberofIOdevicesattachedtothecomputer.
- Theyareall capable of generatingthe interrupt.
- Whentheinterruptis generated frommorethan onedevice,priorityinterrupt systemisused to determinewhich deviceis to beservicedfirst.
- Deviceswithhighspeed transferaregivenhigher priorityandslowdevices aregivenlowerpriority.
- Establishingthe prioritycan bedonein twoways:

- ✓ UsingSoftware
- ✓ UsingHardware

❖ A poolingprocedureis used to identifyhighest priorityin softwaremeans.

PollingProcedure:

- Thereisone commonbranchaddressforallinterrupts.
- Branch address contain the code that polls the interrupt sources in sequence. Thehighest priorityis tested first.
- Theparticular serviceroutineof thehighest prioritydeviceis served.
- Thedisadvantageis thattimerequired topollthem canexceedthetimetoservetheminlargenumberofIO devices.

UsingHardware:

- ❖ Hardwareprioritysystemfunctionasanoverallmanager.Itacceptsinterruptrequestanddeterminethepriorities.
- ❖ Tospeedup theoperation eachinterruptingdeviceshas itsown interruptvector.
- ❖ Nopollingisrequired,all decision areestablishedbyhardwarepriorityinterrupt unit.
- ❖ Itcan be establishedbyserial orparallel connectionofinterruptlines.

SerialorDaisyChainingPriority:

- ✓ Devicewith highestpriorityisplacedfirst.
- ✓ Devicethatwants theattention sendtheinterruptrequestto theCPU.
- ✓ CPU then sends the INTACK signal which is applied to PI(priority in) of the firstdevice.
- ✓ If it had requested the attention, it place its VAD(vector address) on the bus. And itblock thesignal byplacing0in PO(priorityout)
- ✓ If not it passthe signal tonextdevicethrough PO(priorityout) byplacing1.
- ✓ Thisprocessiscontinueduntilappropriatedevice isfound.

- Thedevice whosePIis 1and PO is 0 isthe device that send the interruptrequest.

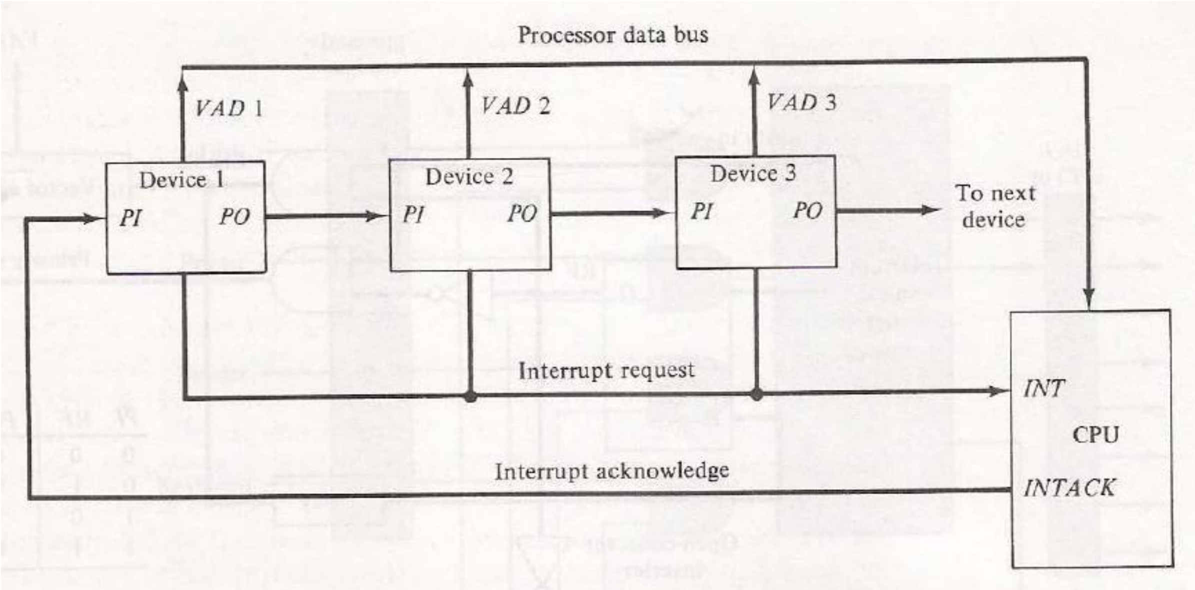
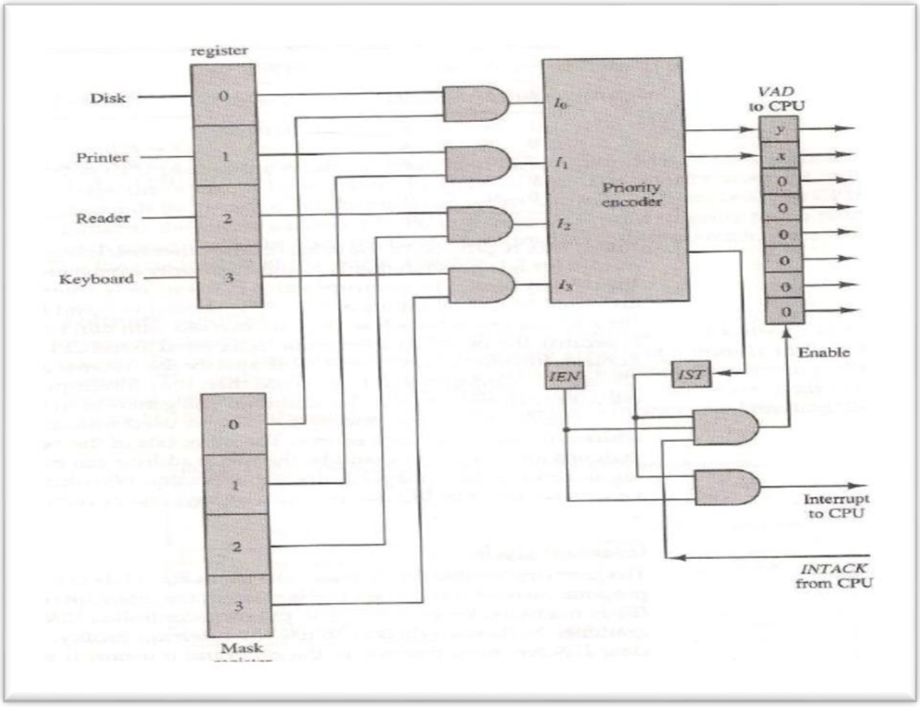


Figure Daisy-chain priority interrupt.

ParallelPriorityInterrupt:

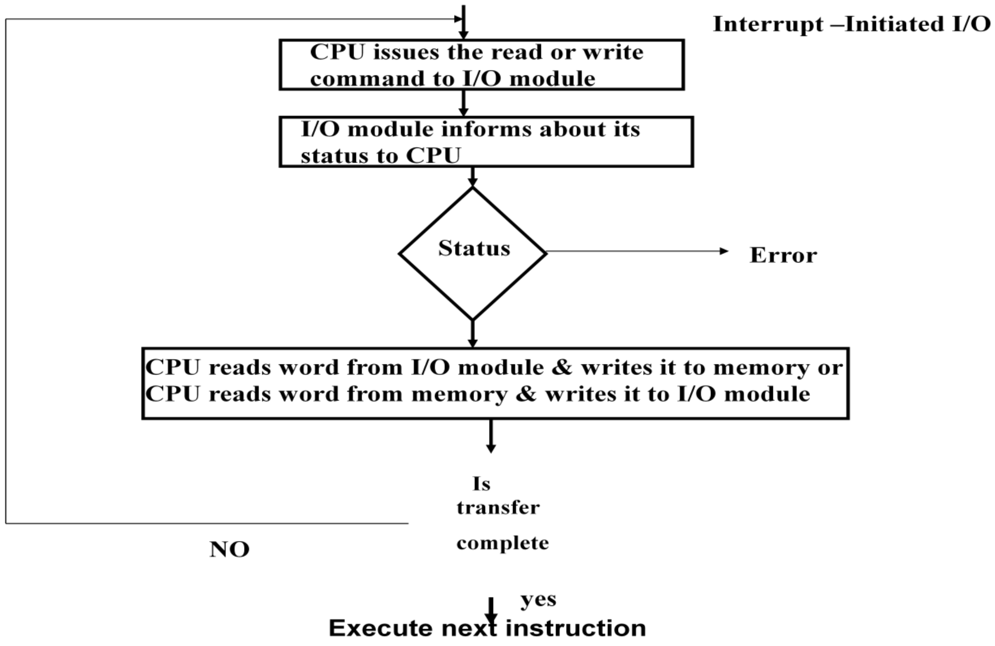
- Itconsist of interrupt registerwhosebits areset separatelybythe interruptingdevices.
- Priorityis establishedaccordingto theposition of the bits intheregister.
- Mask register is used to provide facility for the higher priority devices to interruptwhen lower priority device is being serviced or disable all lower priority deviceswhenhigher is beingserviced.
- Correspondinginterruptbit andmask bitareANDedand appliedto priorityencoder.
- Priorityencodergenerates two bits of vectoraddress.
- AnotheroutputfromitsetsIST(interruptstatusflipflop).



Priority Encoder Truth Table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I'_0 I'_1$
0	0	1	×	1	0	1	$y = I'_0 I_1 + I'_0 I'_2$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

TheExecutionprocess ofInterrupt–InitiatedI/Oisrepresentedintheflowchart:



4.3.3. DirectMemoryAccess(DMA):

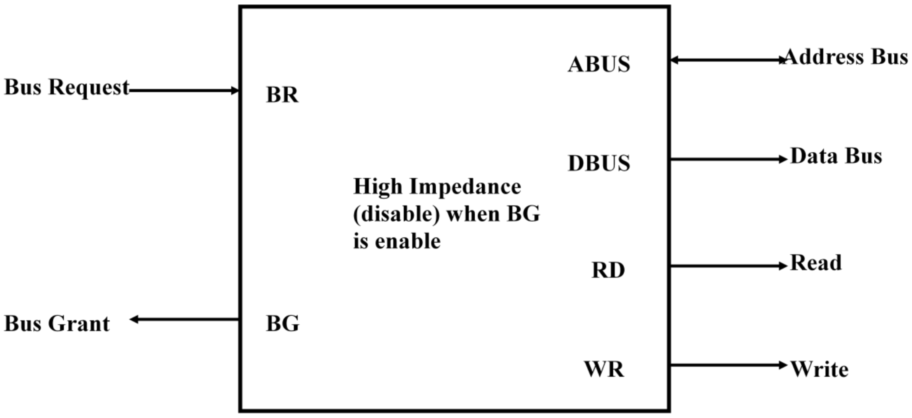
In the Direct Memory Access (DMA) the interface transfer the data into and out of thememoryunit through the memorybus. Thetransferofdata betweenafast storagedevicesuchas magnetic disk and memory is often limited by the speed of the CPU. Removing the CPUfrom the path and letting the peripheral device manage the memory buses directly wouldimprove the speed of transfer. This transfer technique is called Direct Memory Access(DMA).

During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMAController takes over the buses to manage the transfer directly between the I/O device andmemory.

The CPU may be placed in an idle state in a variety of ways. One common methodextensively used in microprocessor is to disable the buses through special control signalssuchas:

- BusRequest(BR)
- BusGrant(BG)

These two control signals in the CPU that facilitates the DMA transfer. The *Bus Request(BR)* input is used by the *DMA controller* to request the CPU. When this input is active, theCPUterminatestheexecutionofthecurrentinstructionandplacesthe addressbus,databusand read write lines into a *high Impedance state*. High Impedance state means that the outputisdisconnected.



CPU bus Signals for DMA Transfer

The CPU activates the *Bus Grant (BG)* output to inform the external DMA that the BusRequest (BR) can now take control of the buses to conduct memory transfer withoutprocessor.

When the DMA terminates the transfer, it disables the *Bus Request (BR)* line. The CPUdisablesthe*BusGrant(BG)*,takescontrolof the busesandreturn toitsnormaloperation.

Thetransfercanbemade inseveral waysthatare:

i. DMABurst

ii. CycleStealing

i) ***DMA Burst :-*** In DMA Burst transfer, a block sequenceconsisting of a number ofmemory words is transferred in continuous burst while the DMA controller is masterofthe memorybuses.

ii) ***CycleStealing:-***CyclestealingallowstheDMAcontroller totransferonedatawordata time, afterwhich it mustreturns control of thebuses totheCPU.

DMAController:

The DMA controller needs the usual circuits of an interface to communicate with theCPUandI/O device.TheDMAcontroller hasthreeregisters:

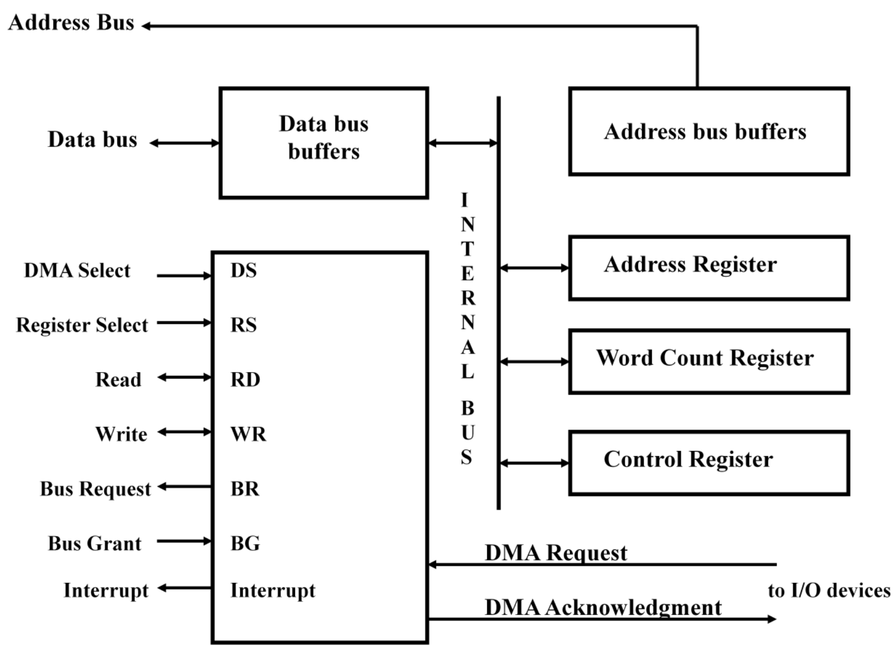
i. AddressRegister
ii. WordCountRegister
iii. ControlRegister

i. ***AddressRegister:-***AddressRegistercontainsanaddresstospecifythedesiredlocationin memory.

ii. ***Word Count Register :-*** WC holds the number of words to be transferred. Theregisteris incre/decrebyoneaftereach wordtransfer and internallytested forzero.

iii. ***ControlRegister:-***ControlRegisterspecifiesthe modeoftransfer

The unit communicates with the CPU via the data bus and control lines. Theregistersin theDMAareselected bytheCPU throughtheaddressbus byenablingtheDS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write)inputsarebidirectional.



Block Diagram of DMA Controller

When the BG (Bus Grant) input is 0, the CPU can communicatewith the DMA registers through the data bus to read from or write to the DMAregisters. When BG =1, the DMA can communicate directly with the memory byspecifyingan address inthe address busand activatingthe RD orWR control.

DMA Transfer:

The CPU communicates with the DMA through the address and data buses as withany interface unit. The DMA has its own address, which activates the DS and RSlines. The CPU initializes the DMA through the data bus. Once the DMA receives thestartcontrol command,it cantransfer betweentheperipheraland thememory.

When BG = 0 the RD and WR are input lines allowing the CPU tocommunicate with the internal DMA registers. When BG=1, the RD and WR areoutput lines from the DMA controller to the random access memory to specify thereadorwriteoperation ofdata.

Summary:

- ❖ Interface is the point where a connection is made between two different parts of asystem.
- ❖ The strobe control method of Asynchronous data transfer employs a single controllineto timeeach transfer.
- ❖ The handshaking method solves the problem of strobe method by introducing asecondcontrol signal that providesareplyto theunitthat initiates thetransfer.
- ❖ Programmed I/Omodeofdatatransfertheoperationsaretheresultsin I/Oinstructionswhichis a part ofcomputer program.
- ❖ In the Interrupt Initiated I/O method an interrupt facility an interrupt command is usedtoinform thedevice about thestart and endoftransfer.
- ❖ In the Direct Memory Access (DMA) the interface transfer the data into and out of thememoryunit through thememorybus.

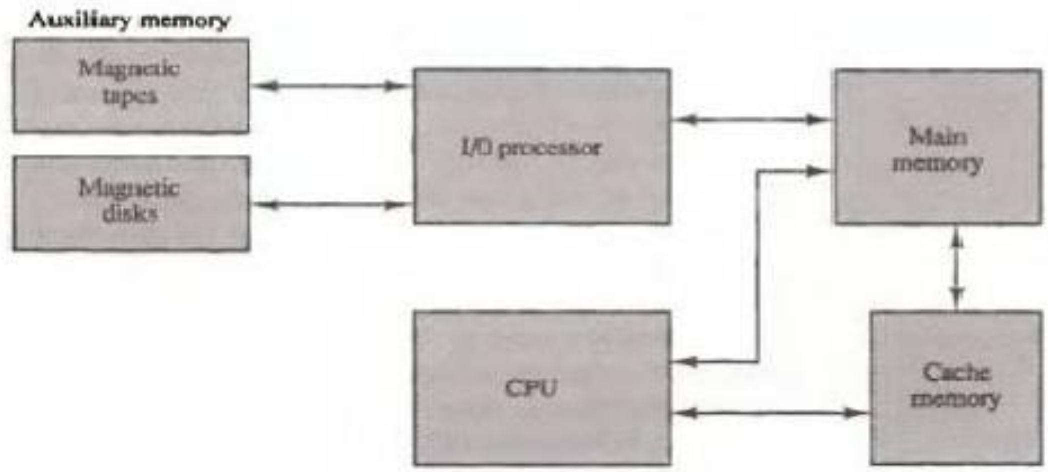
UNIT – 04 – Part - B

MEMORY ORGANIZATION

4.4. Memory Hierarchy:

Memory hierarchy system consists of all storage devices employed in a computer systemfrom the slow but high capacity auxiliary memory to a relatively faster main memory, to an even smallerandfaster cachememoryaccessibletothe highspeed processing logic.

- **MainMemory**:memoryunitthatcommunicatesdirectlywiththeCPU(RAM)
- **AuxiliaryMemory**:devicethatprovidebackupstorage(DiskDrives)
- **CacheMemory**:specialvery-high-speedmemory toincreasetheprocessingspeed(CacheRAM)



Above figure illustrates the components in a typical memory hierarchy. At the bottomof thehierarchy arethe relatively slow magnetictapesused to storeremovable files.Next are theMagneticdisksusedasbackupstorage.Themainmemoryoccupiesacentralpositionbybeingabletocom municate directly with CPU and with auxiliary memory devices through an I/O process. Program notcurrentlyneededinmainmemoryaretransferredintoauxiliarymemorytoprovidespaceforcurrentlyus edprogramsanddata.

The cache memory is used for storing segments of programs currently being executed inthe CPU. The I/O processor manages data transfer between auxiliary memory and main memory. Theauxiliary memory has a large storage capacity is relatively inexpensive, but has low access speedcompared to main memory. The cache memory is very small, relatively expensive, and has very highaccessspeed.TheCPUhas directaccesstobothcacheandmainmemorybutnotto auxiliarymemory.

Multiprogramming:

ManyoperatingsystemsaredesignedtoenabletheCPUtoprocessanumberofindependentprogra ms concurrently.

Multiprogrammingreferstotheexistenceof2ormoreprogramsindifferentpartsofthememoryhierarchyatthesametime.

MemorymanagementSystem:

Thepartofthecomputersystemthatsupervisestheflowofinformationbetweenauxiliarymemoryandmainmemory.

4.5. MainMemory:

Main memory is the central storage unit in a computer system. It is a relatively large andfast memory used to store programs and data during the computer operation. The principal technologyused for the main memory is based on semi conductor integrated circuits. Integrated circuits RAM chipsareavailableintwo possibleoperatingmodes,staticanddynamic.

- ***StaticRAM***–Consistsofinternalflipflopsthatstorethebinaryinformation.
- ***DynamicRAM***– Storesthebinaryinformationintheformofelectricchargesthatareappliedtocapacitors.

MostofthemainmemoryinageneralpurposecomputerismadeupofRAMintegratedcircuitchips,buta portionofthe memorymaybeconstructedwithROMchips.

Read Only Memory –Store programs that are permanently resident in the computer and fortables of constants that do not change invalue once the production of thecomputer iscompleted.

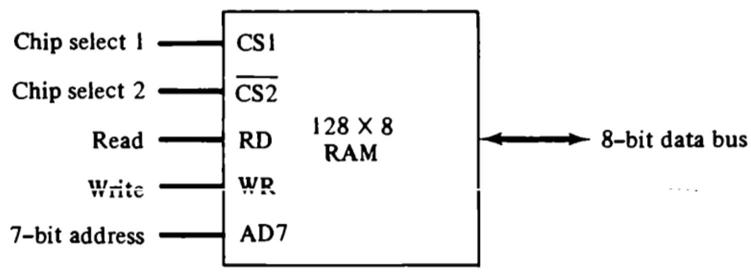
.

TheROMportionofmainmemoryisneededforstoringaninitialprogramcalledBootstrap loader.

- Bootstraploader–functionisstartthecomputersoftwareoperatingwhenpoweristurnedon.
- Bootstrapprogramloadsaportionofoperatingsystemfromdisctomainmemory andcontrolis thentransferredtooperatingsystem.

RAMandROM CHIP:

RAMchip–utilizesbidirectionaldatabuswiththreestatebufferstoperformcommunicationwith CPU.



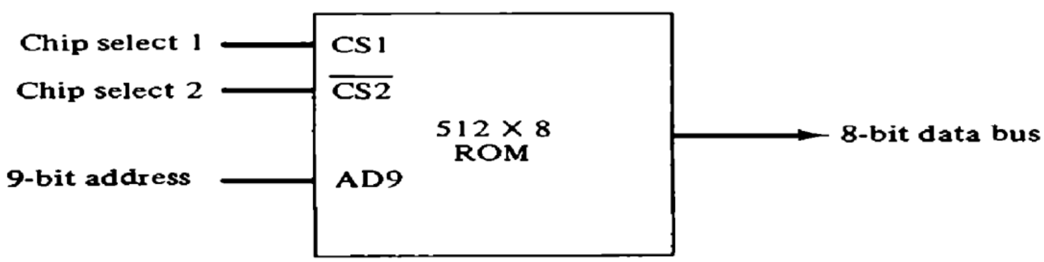
(a) Block diagram

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

(b) Function table

The block diagram of a RAM Chip is shown in Fig. The capacity of memory is 128 words ofeight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus. The readand write inputs specify the memory operation and the two chips select (CS) control inputs are enablingthe chip only when it is selected by the microprocessor. The read and write inputs are sometimescombined intoonelinelabelledR/W.

The function table listed in Fig.(b) specifies the operation of the RAM chip. The unit is inoperation only when CS1=1 and CS2=0.The bar on top of the second select variable indicates that thisinput is enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled butthe read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedancestate.WhenCS1=1andCS2=0,thememory canbeplacedinawriteorreadmode.Whenthe WRinputisenabled,thememorystoresabytefromthedatabusintoalocationspecifiedbytheaddressinputlines. When the RD input is enabled, the content of the selected byte is placed into the data bus. The RDand WR signals control the memory operation as well as the bus buffers associated with the bidirectionaldata bus.



A ROM chip is organized externally in a similar manner. However, sincea ROM can only read,the data bus can only be in an output mode. The block diagram of a ROM chip is shown in fig.12-3.

ThenineaddresslinesintheROMchipspecifyanyoneofthe512bytesstoredinit.Thetwochipselectinputs must be CS1=1 and CS2=0 for the unit to operate. Otherwise, the da~~ta bu~~s is in a high-impedancestate.

Component	Hexadecimal address	Address bus 10	9	8	7	6	5	4	3	2	1
RAM 1	0000–007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080–00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100–017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180–01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200–03FF	1	x	x	x	x	x	x	x	x	x

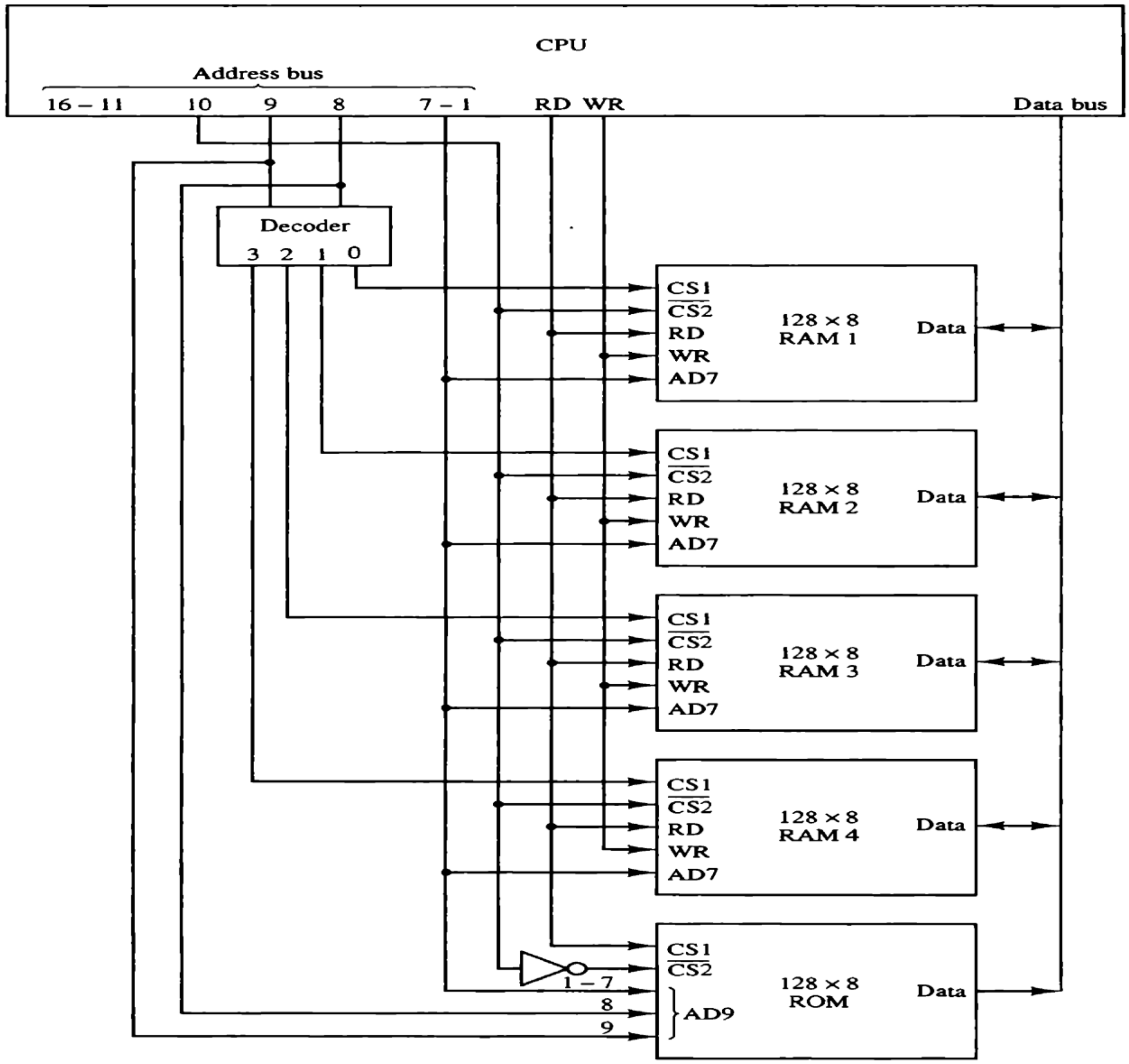
MemoryAddressMap:

The interconnection between memory and processor is then established from knowledge of thesizeofmemoryneededandthetypeofRAMandROMchipsavailable.Theaddressingofmemorycanbe established by means of a table that specify the memory address assigned to each chip. The tablecalled Memory address map, is a pictorial representation of assigned address space for each chip in thesystem.

The memory address map for this configuration is shown in table 12-1. The component columnspecifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range ofhexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. TheRAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9address lines.

MemoryConnectiontoCPU:

RAMandROM chipsareconnected toa CPU through the dataand addressbuses. The loworder linesin the address bus select the byte within the chips and other lines in the address bus select aparticularchipthroughitschipselectinputs.



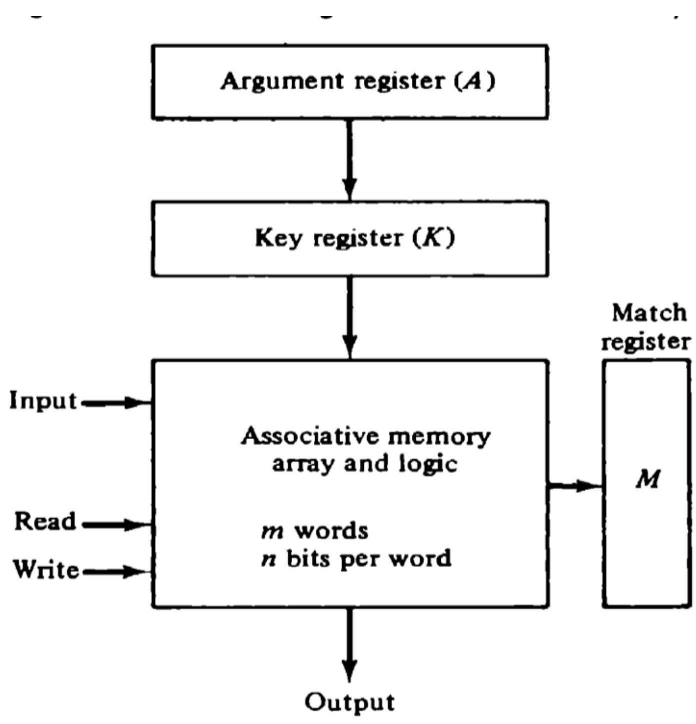
The connection of memory chips to the CPU is shown in Fig.12-4. This configuration gives amemory capacity of 512 bytes of RAM and 512 bytes of ROM. Each RAM receives the seven low-orderbitsoftheaddressbustoselectoneof128possiblebytes.TheparticularRAMchipselectedisdetermined from lines 8 and 9 in the address bus. This is done through a 2 X 4 decoder whose outputs goto the CS1 inputs in each RAM chip. Thus, when address lines 8 and 9 are equal to 00, the first RAM chipis selected. When 01, the second RAM chip is select, and so on. The RD and WR outputs from themicroprocessor are applied to the inputs of each RAM chip. The selection between RAM and ROM isachievedthroughbusline10.TheRAMsareselectedwhenthebitinthislineis0,andtheROMwhenthe bit is 1. Address bus lines 1 to 9 are applied to the input address of ROM without going through thedecoder.

The data bus of the ROM has only an output capability, whereas the data bus connected to theRAMs cantransfer information inbothdirections.

4.6. AuxiliaryMemory:

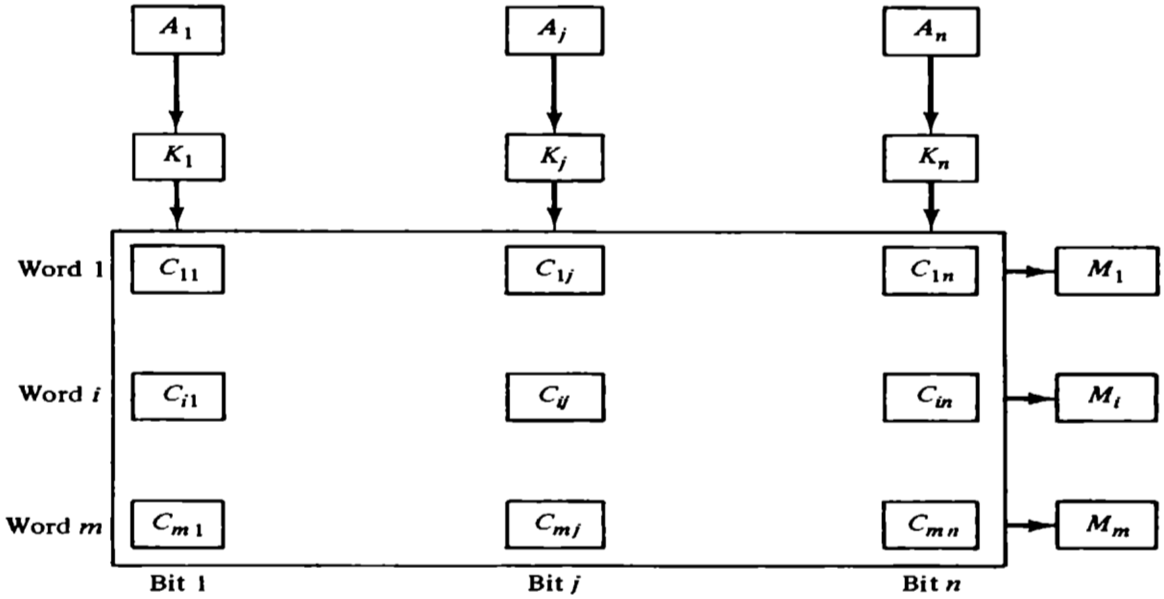
The time required to find an item stored in memory can be reduced considerably if stored data can beidentified for access by the content of the data itself rather than by an address. A memory unit accessedbycontentis calledanassociativememoryor content addressablememory(CAM).

- CAMisaccessedsimultaneouslyandinparallelonthebasisofdatacontentratherthanbys pecific addressor location
- AssociativememoryismoreexpensivethanaRAMbecauseeachcellmusthavestorageca pabilityaswellaslogiccircuits
- Argumentregister–holdsanexternalargumentforcontentmatching
- Keyregister–maskforchoosingaparticularfieldorkeyintheargumentword



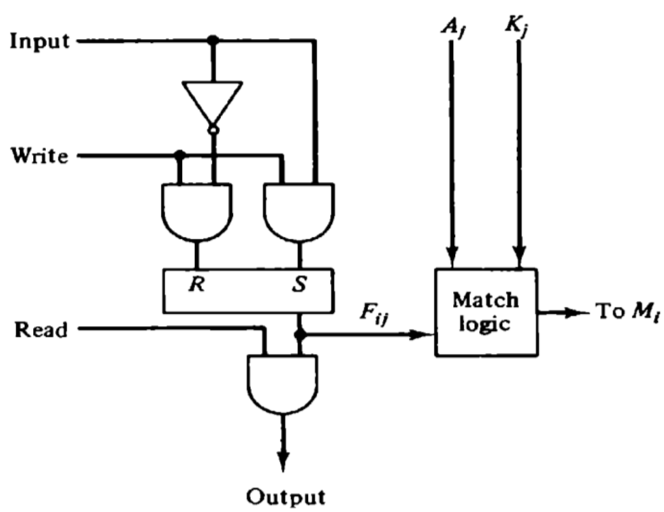
HardwareOrganization

It consists of a memory array and logic for m words with n bits per word. Theargument register A and key register K each have n bits, one for each bit of a word. The match register Mhas m bits, one for each memory word. Each word in memory is compared in parallel with the content ofthe argument register. The words that match the bits of the argument register set a corresponding bit inthe match register. After the matching process, those bits in the match register that have been setindicate the fact that their corresponding words have been matched. Reading is accomplished by asequential access to memory for those words whose corresponding bits in the match register have beenset.



The relation between the memory array and external registers in an associative memory is shownin Fig. The cells in the array are marked by the letter C with two subscripts. The first subscript givesthewordnumberandsecondspecifiesthebitpositionintheword.ThuscellC$_{ij}$isthecellforbitjinw ordi.

AbitA$_j$intheargumentregisteriscomparedwithallthebitsincolumnjofthearrayprovidedthatk$_j$=1.T his is done for all columns j=1,2,....n. If a match occurs between all the unmasked bits of



theargument and the bits in word I, the corresponding bit M_i in the match register is set to 1. If one or moreunmaskedbits of the argumentandtheword do not match,$M_i$is clearedto 0.

Itconsistsofflip-flopstorageelementF_{ij}andthecircuitsforreading,writing,andmatchingthecell. The input bit is transferred into the storage cell during a write operation. The bit stored is read outduring a read operation. The match logic compares the content of the storage cell with correspondingunmaskedbitof theargumentand providesanoutput forthedecision logicthatsets thebitin M_i.

MatchLogic

The match logic for each word can be derived from the comparison algorithm for two binary numbers.First,neglectthekeybitsandcomparethe argumentinAwith thebitsstoredinthecellsofthewords.

Word i is equal to the argument in A if $A_j=F_{ij}$for j=1,2,.....,n. Two bits are equal if they are both 1orboth 0.Theequalityoftwobitscanbeexpressedlogicallybythe Booleanfunction

$$x_j=A_jF_{ij}+A_j'F_{ij}'$$

wherex_j = 1 if the pair of bits in position j are equal;otherwise , x_j =0. For a word i is equal to theargument in A we must have all x_j variables equal to 1. This is the condition for setting the correspondingmatchbit$M_i$to1.The Booleanfunctionforthisconditionis

$$M_i=x_1x_2x_3......x_n$$

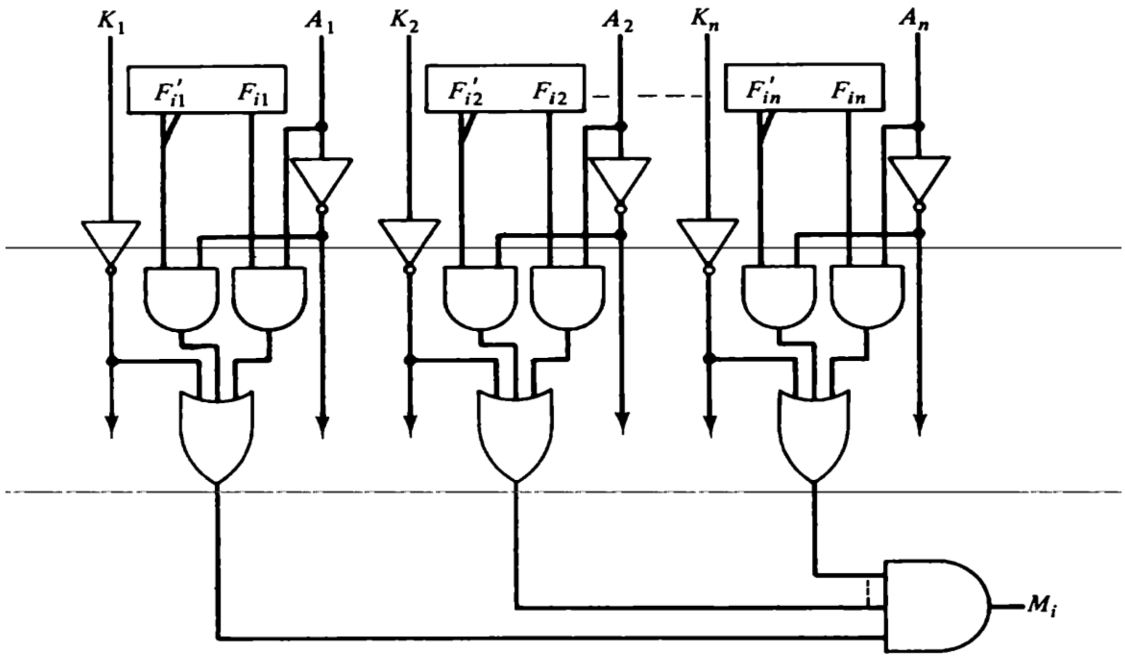


Figure 12-9 Match logic for one word of associative memory.

Each cell requires two AND gate and one OR gate. The inverters for A and K are needed once for eachcolumn and are used for all bits in the column. The output of all OR gates in the cells of the same word goto theinput ofacommon ANDgatetogenerate thematch signal for M_i. $M_i$will belogic 1ifa matchoccurs and0ifnomatchoccurs.

ReadOperation:

Ifmorethanonewordinmemorymatchestheunmaskedargumentfield,allthematchedwordswi llhave1's inthecorrespondingbit positionof thematchregister

- Inreadoperationallmatchedwordsarereadin☐sequencebyapplyingareadsig naltoeachwordlinewhosecorrespondingMibit isalogic1
- Inapplicationswherenotwoidenticalitemsarestoredinthememory,only onewordmaymatch,inwhichcasewecanuseMioutputdirectlyasareadsignalforthecorr espondingword

WriteOperation

Cantaketwodifferentforms;

1. Entire memory may be loaded with new information
2. Unwantedwordstobedeletedandnewwordstobeinserted

1. Entire memory : writing can be done by addressing each location in sequence – This makes it randomaccess memory for writing and content addressable memory for reading – number of lines needed fordecoding isdWherem=2d,misnumber of words.
2. Unwantedwordstobedeletedandnew□wordstobeinserted:
   - ❖ Tagregisteris usedwhichhasas manybits astherearewordsinmemory
   - ❖ Foreveryactive(valid)wordinmemory,thecorrespondingbitintagregisterissetto1
   - ❖ Whenwordisdeletedthecorrespondingtagbitisresetto0
   - ❖ Thewordisstoredinthememorybyscanningthetagregisteruntilthefirst0bitisencount eredAfterstoringthe wordthe bitis setto 1.

4.7. CacheMemory:

Effectiveness of cache mechanism is based on a property of computer programs called **“localityofreference”**

Thereferencestomemoryatanygiventimeintervaltendtobe□confinedwithinalocalizedareas

Analysis of programs shows that most of their execution time is spent on routines in whichinstructions are executed repeatedlyThese instructions may be – loops, nested loops , or fewprocedures thatcalleachother.

Manyinstructionsinlocalizedareasofprogram areexecutedrepeatedly during some time period and reminder of the program is accessed infrequentlyThispropertyiscalled“LocalityofReference”.

LocalityofReference

Localityofreferenceismanifestedintwoways:

1. Temporal-meansthatarecentlyexecutedinstructionislikelytobeexecutedagainverysoon.

   The information which will be used in near future is likely to□ be in use already(e.g. reuseof informationinloops)
2. Spatial- means that instructions in close proximity to a recently executed instruction are

also likelytobeexecutedsoon

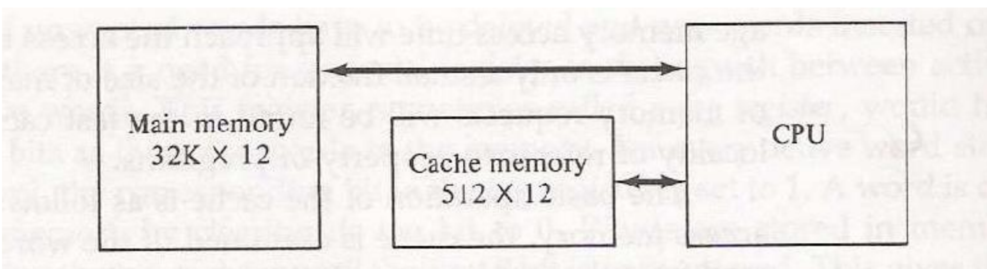
Ifawordisaccessed,adjacent(near)wordsarelikelytobeaccessedsoon(e.g.relatedd ataitems(arrays)areusuallystoredtogether;instructionsareexecutedsequentially)

If active segments of a program can be placed in afast (cache) memory , then total execution timecanbereducedsignificantly

Temporal Locality of Reference suggests whenever an information (instruction or data) is neededfirst,thisitemshouldbebroughtintocache

Spatial aspect of Locality of Reference suggeststhat instead of bringing just one item from themain memory to the cache ,it is wise to bring several items that reside at adjacent addresses aswell(ieablockof information)

Principleofcache:



The main memory can store 32k words of 12 bits each. The cache is capable of storing 512 ofthesewords at any given time. For every word stored , thereis a duplicate copy in main memory. TheCpu communicates with both memories. It first sends a 15 bit address to cahache. If there is a hit, theCPU accepts the 12 bit data from cache. If there is a miss, the CPU reads the word from main memoryandthewordisthentransferredtocache.

- Whenareadrequestisreceivedfrom CPU,contentsofablockofmemorywordsco ntainingthelocationspecifiedaretransferredintocache
- Whentheprogramreferencesanyofthelocationsinthisblock,thecontentsarere adfromthecacheNumber of blocks incacheis smaller thannumberof blocks inmainmemory.

- Correspondencebetweenmainmemoryblocksandthoseinthecacheisspecifiedbyama ppingfunction

- Assumecacheisfullandmemorywordnotincacheisreferenced
- Controlhardwaredecideswhichblockfromcacheistoberemovedtocreatespacefornew blockcontainingreferencedwordfrommemory
- Collection of rules for making this decision is called **“Replacement algorithm ”.**

CacheHitOperation:

- CPUissuesRead/Writerequestsusingaddressesthat□refertolocations inmainmemory
- Cachecontrolcircuitrydetermineswhetherrequestedwordcurrentlyexistsincache
- Ifitdoes,Read/Writeoperationisperformedontheappropriatelocationincache**(Read/WriteHit)**

Read/WriteoperationsoncacheincaseofHit:

- InReadoperationmainmemoryisnotinvolved.
- InWriteoperationtwothingscanhappen.

1. Cacheandmainmemorylocationsareupdated□simultaneously**(“WriteThrough”)**OR

2. Updateonlycachelocationandmarkitas“Dirtyor□ModifiedBit”andupdatemainmemory location atthetimeofcacheblockremoval**(“ WriteBack” or“ CopyBack”)**.

Read/WriteoperationsoncacheincaseofMissReadOperation:

WhenaddressedwordisnotincacheReadMissoccursthereraretwowaysthiscanbedealtwith

1. Entireblockofwordsthatcontaintherequestedwordiscopiedfrommainmemorytocacheand theparticularwordrequestedis forwardedtoCPUfromthecache**(LoadThrough)**(OR)

2. TherequestedwordfrommemoryissenttoCPUfirstandthenthecacheisupdated**(EarlyRestart)**

WriteOperation:

- IfaddressedwordisnotincacheWriteMissoccurs
- Ifwritethroughprotocolisusedinformationisdirectlywrittenintomainmemory

- ✓ Inwritebackprotocol,blockcontainingthewordisfirstbroughtintocache,thedesiredw ordisthen overwritten.

4.8. MappingFunctions:

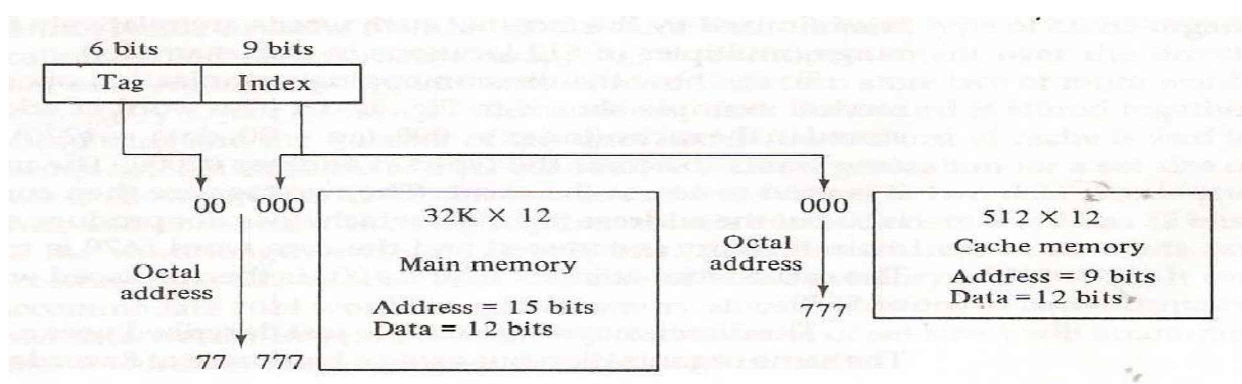
Correspondencebetweenmainmemoryblocksandthoseinthecacheisspecifiedbyame morymappingfunction

Therearethreetechniques inmemorymapping

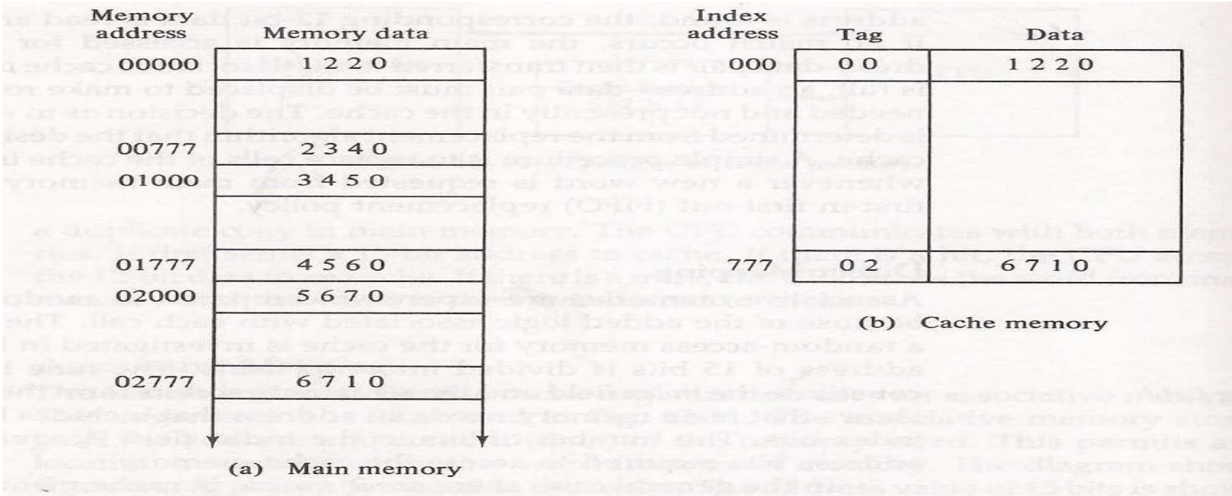
- ❖ DirectMapping
- ❖ AssociativeMapping
- ❖ SetAssociativeMapping

Directmapping:

A particular blockof main memory can be broughtto a particular blockofcache memory.So,itis notflexible.

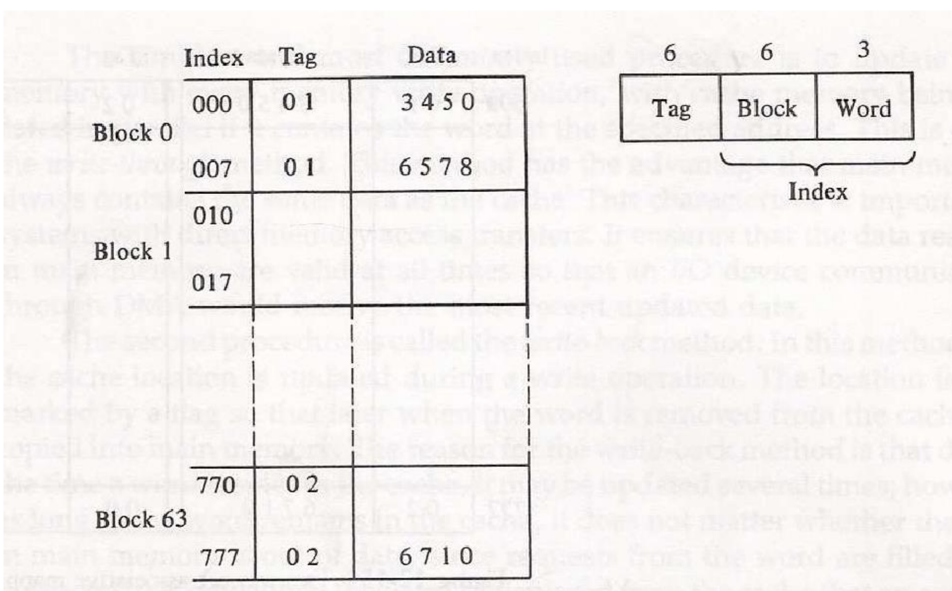


IntheCPUaddressof15bitsisdividedintotwofields.Thenineleastsignificantbitsconsti tutetheindexfieldandremainingsixbitsformthetagfield.Themainmemoryneedsanaddresst hatincludesboththetagandtheindexbits.Thenumberofbitsintheindexfieldisequaltothenu mberof addressbitsrequiredtoaccessthecachememory.



The direct mapping cache organization uses the n- bit address to access the main memory andthe k-bit index to access the cache.Each word in cache consists of the data word and associated tag.When a new word is first brought into the cache, the tag bits are stored alongside the data bits.

When theCPU generates a memory request, the index field is used the index field is used for the address toaccess the cache. The tag field of the CPU address is compared with the tag in the word read from thecache. If the two tags match, there is a hit and the desired data word is in cache. If there is no match,there isamissandtherequiredword isread frommainmemory.

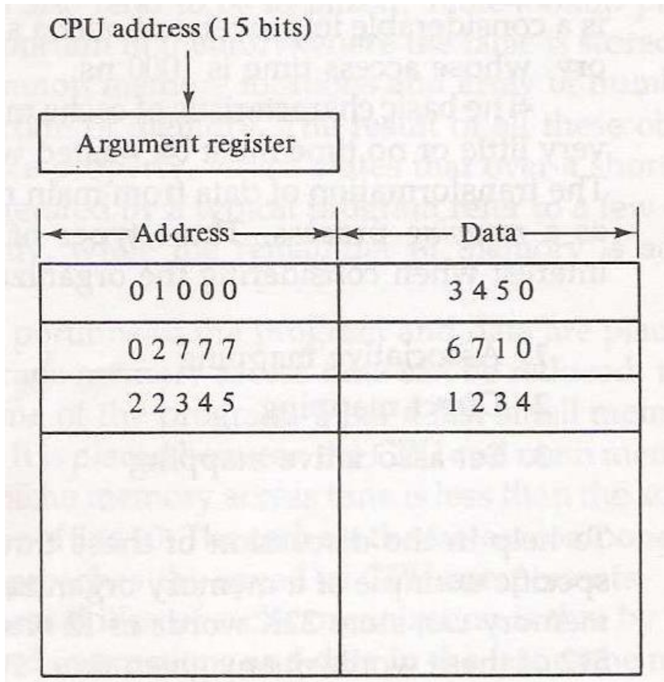


In fig,the index field is now divided into two parts: Block field and The word field.

In a 512word cache there are 64 blocks of 8 words each, since 64X8=512. The block number is specified with a 6bit field and the wordwith in the block is specified with a 3-bit field. Thetag field stored within the cacheiscommontoalleight wordsofthesameblock.

Associativemapping:

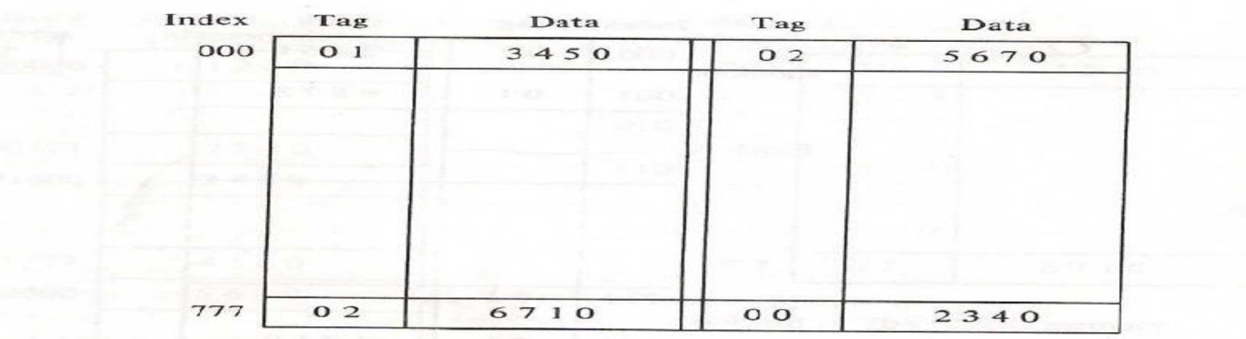
In this mapping function, any block of Main memory can potentially reside in any cache blockposition. Thisismuchmoreflexiblemappingmethod.



In fig, the associative memory stores both address and content(data) of the memory word.This permits any location in cache to store any word from main memory.The diagram shows three wordspresently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and itscorresponding 12-bit word is shown as a four-digit octal number. A CPU address of 15-bits is placed intheargumentregisterandtheassociativememoryissearchedforamatchingaddress.Ifaddr essisfound, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory isaccessedfor theword.

Set-associativemapping:

In this method, blocks of cache are grouped into sets, and the mapping allows a block of mainmemory to reside in any block of a specific set. From the flexibility point of view, it is in between to theothertwomethods.



The octal numbers listed in Fig. are with reference to the mainmemorycontents.WhentheCPU generatesamemoryrequest,theindexvaluesoftheaddressisusedtoaccess thecache.ThetagfieldoftheCPUaddress is then compared with both tags in the cache to determine if a match occurs. The comparisonlogic dine by an associative search of the tags in the set similar to anassociative memory search thus thename“SetAssociative”.

ReplacementPolicies:

- ❖ When the cache is full and there is necessity to bring new data to cache , then a decision must bemadeastowhichdatafromcacheis toberemoved
- ❖ The guideline for taking a decision about which data is to be removed is called replacement policyReplacementpolicydependsonmapping
- ❖ ThereisnospecificpolicyincaseofDirectmappingaswehavenochoiceofblock placementincacheReplacementPolicies

Incaseofassociativemapping

- ➢ Asimpleprocedureistoreplacecellsofthecacheinroundrobinorderwhenever anewwordisrequestedfrommemory
- ➢ ThisconstitutesaFirst-inFirst-out(FIFO)replacementpolicy

Incaseofsetassociativemapping

- ✓ Randomreplacement
- ✓ First-inFirst-out(FIFO)(item chosenistheitemthathasbeeninthesetlongest)
- ✓ LeastRecentlyUsed(LRU)(itemchosenistheitem☐thathasbeenleastrecentl yusedbyCPU)

4.9. VirtualMemory:

- Earlydays memorywas expensive–hencesmall
- Programmerswereusingsecondarystorageforoverlaying
- Programmerswereresponsibleforbreakingprogramsintooverlays,decidewheretokeepinsecondary memory,arrangingfortransferofoverlaysbetweenmainmemoryandsecondarymemory.

In1961ManchesterUniversityproposedamethodforperformingoverlayprocessautomaticallywhichhasgivenrisetotheconceptofVirtual memorytoday.

VirtualMemory-Background

- Separateconceptofaddressspaceand□memorylocations
- ProgramsreferenceinstructionsanddatathatisindependentofavailablephysicalmemoryAddresses issued by processor for Instructions or Data are called Virtual orLogical addresses
- Virtual addresses are translated in to physical□ addresses by a combination of Hardwareand Softwarecomponents

TypesofMemory

- ✓ Real memory
- ✓ Mainmemory
- ✓ Virtualmemory
- ✓ Memoryondisk

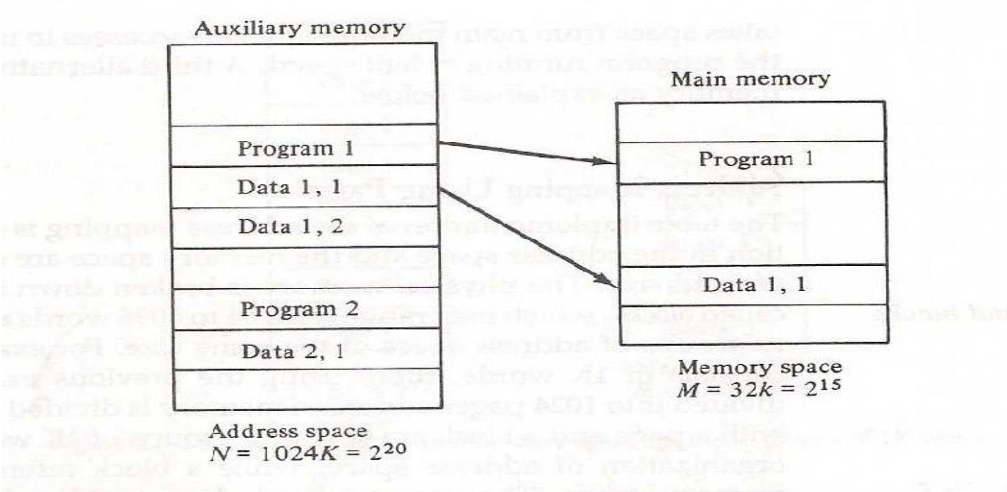
Allowsforeffectivemultiprogrammingand□relievestheuseroftightconstraints ofmainmemory.

AddressSpaceandMemorySpace

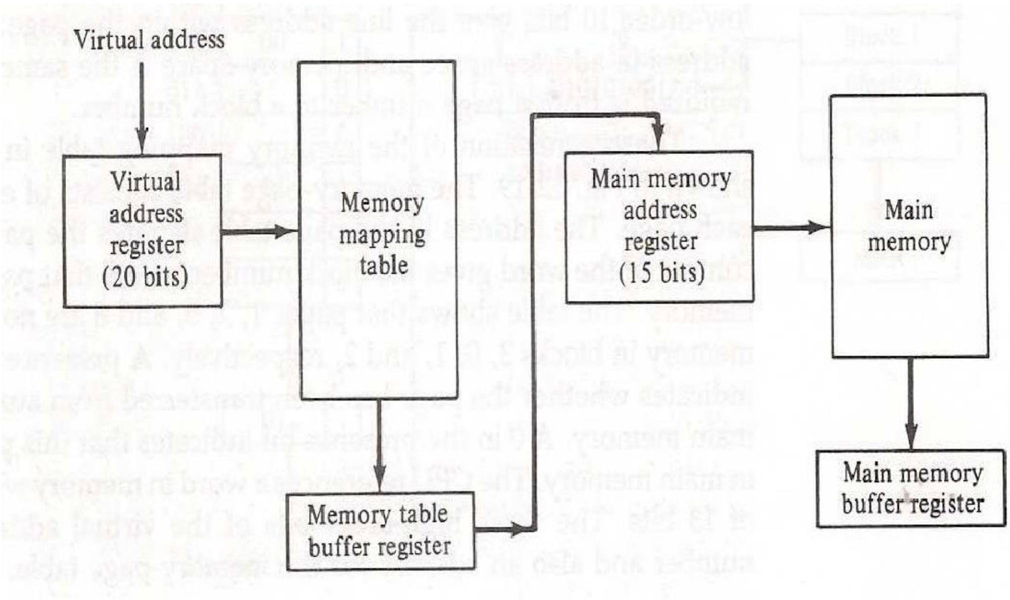
- Addressusedbyaprogrammeriscalledvirtualaddressandsetofsuchaddressesiscalled addressspace.

- Addressinmainmemoryiscalledalocation□orphysicaladdressandsetofsuchlocationsiscalledthememoryspace.

❖ TheAddressSpaceisallowedtobelarger□thanthememoryspaceincomputerswithvir tualmemory.



Inamultiprogramcomputersystem,programsanddataaretransferredtoandfromau xiliarymemoryandmainmemorybasedondemandsimposedbytheCPU.Supposethatprogr am1iscurrentlybeing executed in the CPU. Program1 and a portion of its associated data are moved from auxiliarymemory into main memory as shown in fig. Portions of programs and data need not be incontiguous locations in memory since information is

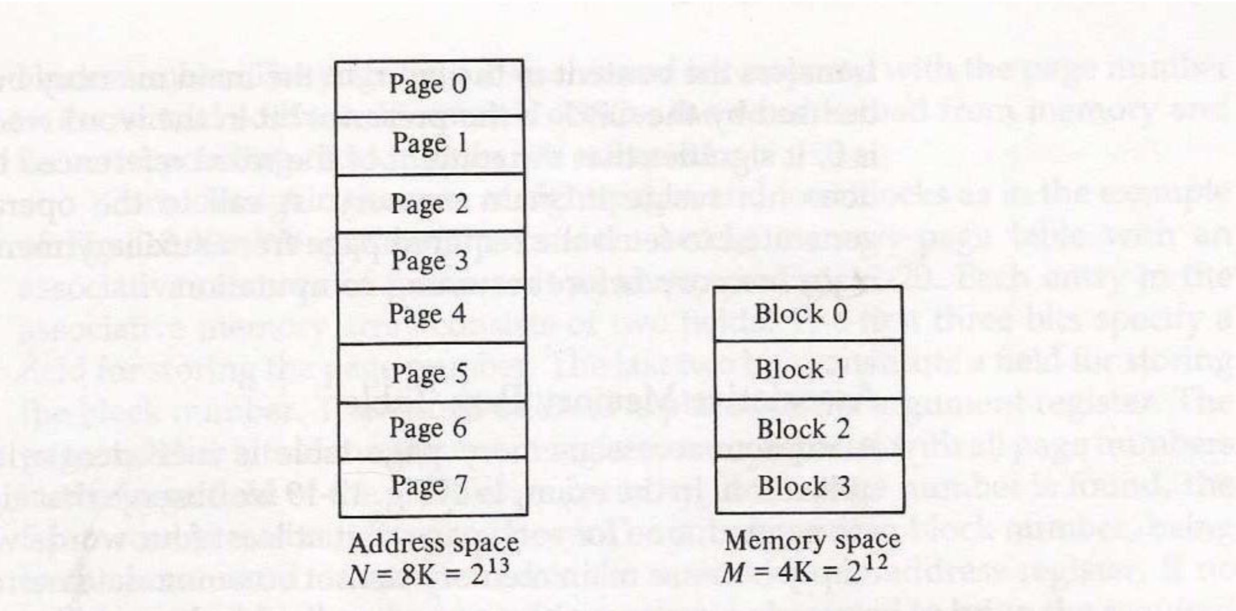


being moved in out, and empty spaces may beavailable inscatteredlocationsin memory.

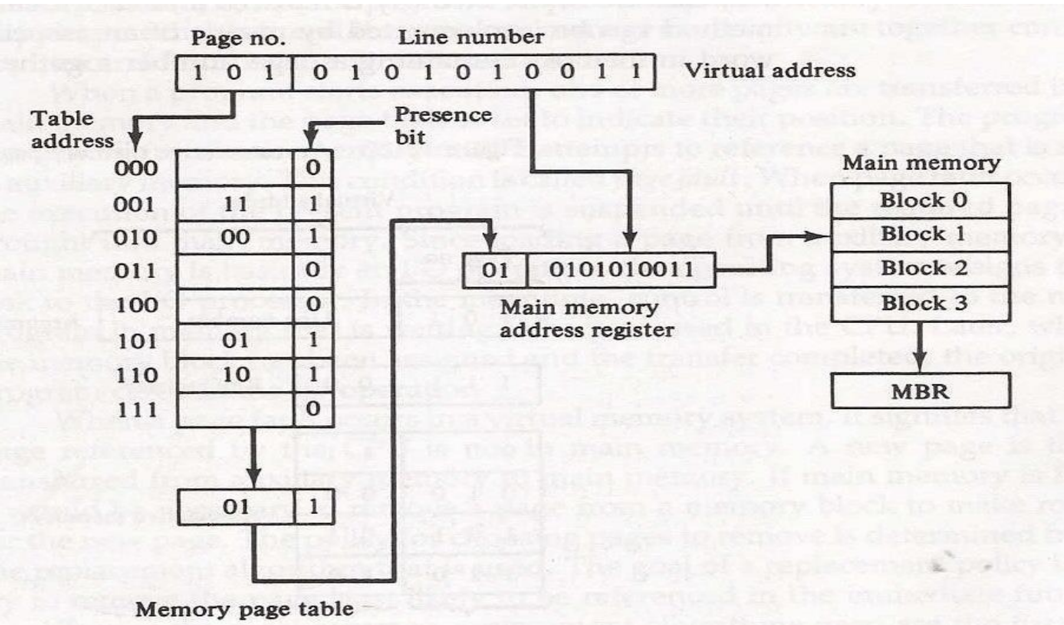
In fig, to map a virtual address of 20 bits to a physical address of 15 bits. The mapping is adynamic operation, which means that every address is translated immediately as a word is referenced byCPU.The mapping table may be stored in a separate memory. In first case, an additional unit is requiredas well as one extra memory access time. In the second case, the table takes space from main memoryand two accesses to memory are required with program running at half speed. A third alternative is to useanassociativememory.

AddressMappingUsingPages

The physical memory is broken down into groups of equal size called blocks, which may rangefrom64to4096wordeach.Thetermpagereferstogroupsofaddressspaceofthesamesize.Portionsofprograms are moved fromauxiliary memory to main memory in records equal to the sizeofa page.Theterm“pageframe” issometimes usedtodenoteablock.



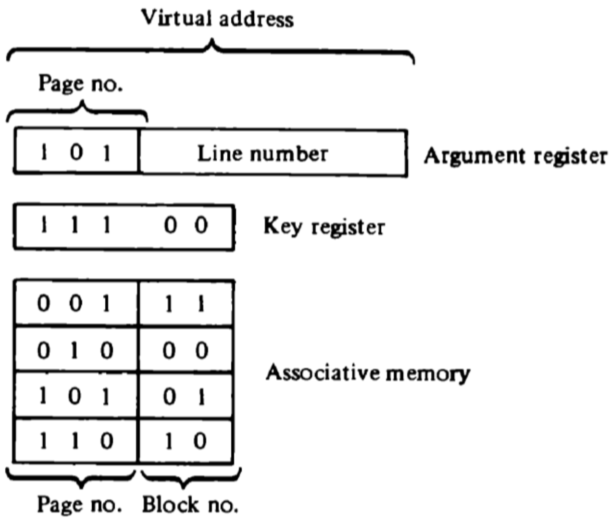
In fig, a virtual address has 13 bits. Since each page consists of 1024 words, the high orderthree bits of virtual address will specify one of the eight pages and the low order 10 bits give the lineaddresswithinthepage.



The organizationof the memory mapping table in a paged system is shown in Fig.12-19. Thememory page table consists of eight word , one for each page. The address in the page tabledenotes thepagenumberandthecontentofthewordgivestheblocknumberwherethatpageisstoredin mainmemory. The table showsthat pages 1,2,5 and 6 are now available in main memory in blocks 3,0,1 and 2,respectively.

AssociativeMemoryPageTable

Arandom-accessmemorypagetableisinefficientwithrespecttostorageutilization.



Replace the random access memory-page table with an associative memory of four words asshown in Fig12-20. Each entry in the associative memory array consists of two fields. The first three bitsspecifyafieldforstoringthepagenumber.Thelasttwobitsconstituteafieldforstoringthe blocknumber.Thevirtualaddress isplacedintheargumentregister.

AddressTranslation

Atableisneededtomapvirtualaddresstoaphysicaladdress(dynamicoperation)This tablemaybekeptin

- aseparatememoryor
- mainmemoryor
- associativememory

UNIT – 05 – Part - A

REDUCED INSTRUCTION SET COMPUTER

CISC Processor:

The CISC Stands for **Complex Instruction Set Computer**, developed by the Intel. It has a large collection of complex instructions that range from simple to very complex and specialized in the assembly language level, which takes a long time to execute the instructions. So, CISC approaches reducing the number of instruction on each program and ignoring the number of cycles per instruction. It emphasizes to build complex instructions directly in the hardware because the hardware is always faster than software. However, CISC chips are relatively slower as compared to RISC chips but use little instruction than RISC. Examples of CISC processors are VAX, AMD, Intel x86 and the System/360.

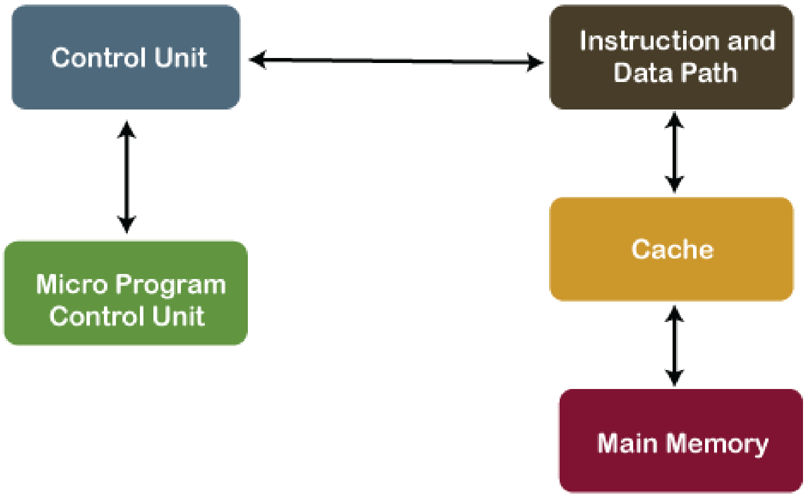
Characteristics of CISC Processor:

Following are the main characteristics of the RISC processor:

1. The length of the code is shorts, so it requires very little RAM.
2. CISC or complex instructions may take longer than a single clock cycle to execute the code.
3. Less instruction is needed to write an application.
4. It provides easier programming in assembly language.
5. Support for complex data structure and easy compilation of high-level languages.
6. It is composed of fewer registers and more addressing nodes, typically 5 to 20.
7. Instructions can be larger than a single word.
8. It emphasizes the building of instruction on hardware because it is faster to create than the software.

CISC Processors Architecture:

The CISC architecture helps reduce program code by embedding multiple operations on each program instruction, which makes the CISC processor more complex. The CISC architecture-based computer is designed to decrease memory costs because large programs or instruction required large memory space to store the data, thus increasing the memory requirement, and a large collection of memory increases the memory cost, which makes them more expensive.



CISC Architecture

Advantages of CISC Processors:

1. The compiler requires little effort to translate high-level programs or statement languages into assembly or machine language in CISC processors.
2. The code length is quite short, which minimizes the memory requirement.
3. To store the instruction on each CISC, it requires very less RAM.
4. Execution of a single instruction requires several low-level tasks.
5. CISC creates a process to manage power usage that adjusts clock speed and voltage.
6. It uses fewer instructions set to perform the same instruction as the RISC.

Disadvantages of CISC Processors:

1. CISC chips are slower than RSIC chips to execute per instruction cycle on each program.
2. The performance of the machine decreases due to the slowness of the clock speed.
3. Executing the pipeline in the CISC processor makes it complicated to use.
4. The CISC chips require more transistors as compared to RISC design.
5. In CISC it uses only 20% of existing instructions in a programming event.

RISC Processor:

RISC stands for **Reduced Instruction Set Computer Processor**, a microprocessor architecture with a simple collection and highly customized set of instructions. It is built to minimize the instruction execution time by optimizing and limiting the number of instructions. It means each instruction cycle requires only one clock cycle, and each cycle contains three parameters: fetch, decode and execute. The RISC processor is also used to perform various complex instructions by combining them into simpler ones. RISC chips require several transistors, making it cheaper to design and reduce the execution time for instruction.

Examples of RISC processors are SUN's SPARC, PowerPC, Microchip PIC processors, RISC-V.

Advantages of RISC Processor:

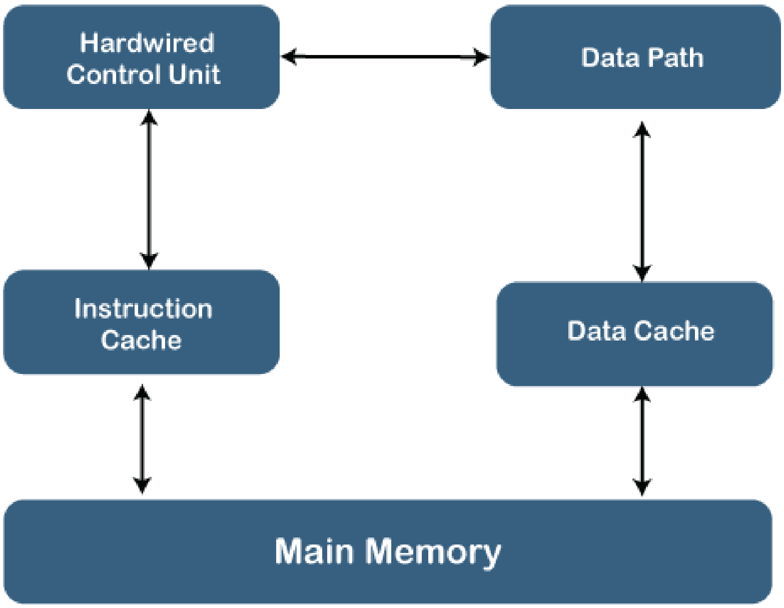
1. The RISC processor's performance is better due to the simple and limited number of the instruction set.
2. It requires several transistors that make it cheaper to design.
3. RISC allows the instruction to use free space on a microprocessor because of its simplicity.
4. RISC processor is simpler than a CISC processor because of its simple and quick design, and it can complete its work in one clock cycle.

Disadvantages of RISC Processor:

1. The RISC processor's performance may vary according to the code executed because subsequent instructions may depend on the previous instruction for their execution in a cycle.
2. Programmers and compilers often use complex instructions.
3. RISC processors require very fast memory to save various instructions that require a large collection of cache memory to respond to the instruction in a short time.

RISC Architecture:

It is a highly customized set of instructions used in portable devices due to system reliability such as Apple iPod, mobiles/smartphones, Nintendo DS,



RISC Architecture

Features of RISC Processor:

Some important features of RISC processors are:

1. **One cycle execution time:** For executing each instruction in a computer, the RISC processors require one CPI (Clock per cycle). And each CPI includes the fetch, decode and execute method applied in computer instruction.
2. **Pipelining technique:** The pipelining technique is used in the RISC processors to execute multiple parts or stages of instructions to perform more efficiently.
3. **A large number of registers:** RISC processors are optimized with multiple registers that can be used to store instruction and quickly respond to the computer and minimize interaction with computer memory.
4. It supports a simple addressing mode and fixed length of instruction for executing the pipeline.
5. It uses LOAD and STORE instruction to access the memory location.
6. Simple and limited instruction reduces the execution time of a process in a RISC.

Difference between the RISC and CISC Processors:

RISC	CISC
It is a Reduced Instruction Set Computer.	It is a Complex Instruction Set Computer.
It emphasizes on software to optimize the instruction set.	It emphasizes on hardware to optimize the instruction set.
It is a hard wired unit of programming in the RISC Processor.	Microprogramming unit in CISC Processor.
It requires multiple register sets to store the instruction.	It requires a single register set to store the instruction.
RISC has simple decoding of instruction.	CISC has complex decoding of instruction.
Uses of the pipeline are simple in RISC.	Uses of the pipeline are difficult in CISC.
It uses a limited number of instruction that requires less time to execute the instructions.	It uses a large number of instruction that requires more time to execute the instructions.
It uses LOAD and STORE that are independent instructions in the register-to-register a program's interaction.	It uses LOAD and STORE instruction in the memory-to-memory interaction of a program.
RISC has more transistors on memory registers.	CISC has transistors to store complex instructions.
The execution time of RISC is very short.	The execution time of CISC is longer.
RISC architecture can be used with high-end applications like telecommunication, image processing, video processing, etc.	CISC architecture can be used with low-end applications like home automation, security system, etc.
It has fixed format instruction.	It has variable format instruction.

The program written for RISC architecture needs to take more space in memory.	Program written for CISC architecture tends to take less space in memory.
Example of RISC: ARM, PA-RISC, Power Architecture, Alpha, AVR, ARC and the SPARC.	Examples of CISC: VAX, Motorola 68000 family, System/360, AMD and the Intel x86 CPUs.

UNIT – 05 – Part -B

PIPELINE AND VECTOR PROCESSING

Parallel processing:

Parallel processing is a term used for a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.

It refers to techniques that are used to provide simultaneous data processing.

The system may have two or more ALUs to be able to execute two or more instruction at the same time. The system may have two or more processors operating concurrently. It can be achieved by having multiple functional units that perform same or different operation simultaneously.

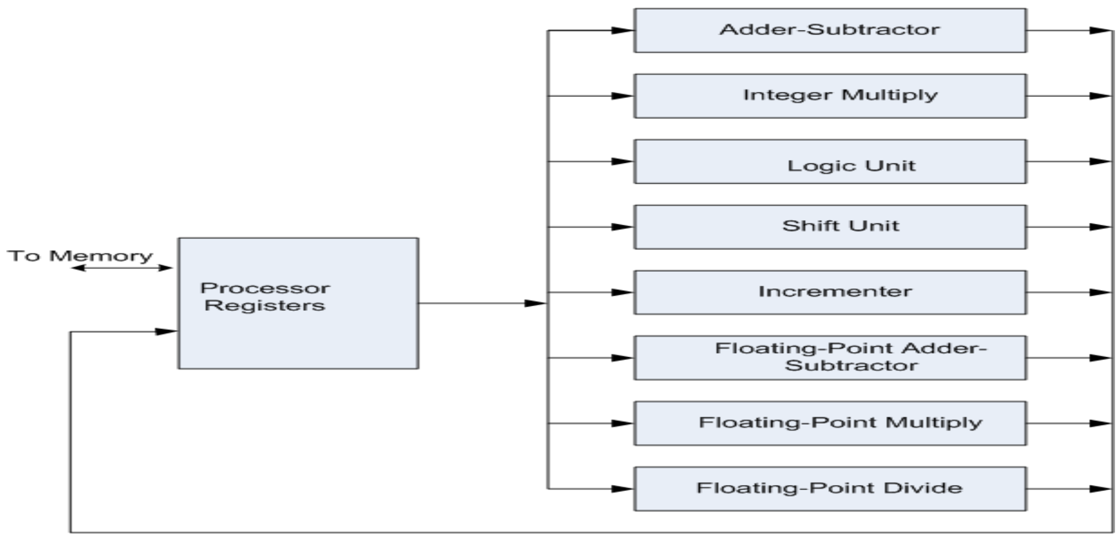
Example of parallel Processing:

Multiple Functional Unit:

Separate the execution unit into eight functional units operating in parallel.

There are variety of ways in which the parallel processing can be classified;

- Internal Organization of Processor
- Interconnection structure between processors
- Flow of information through system



Architectural Classification:

Flynn's classification

- ✓ Based on the multiplicity of *Instruction Streams* and *Data Streams*
- ✓ Instruction Stream
- ✓ Sequence of Instructions read from memory
- ✓ Data Stream
- ✓ Operations performed on the data in the processor

		Number of *Data Streams*	
		Single	Multiple
Number of *Instruction Streams*	Single	SISD	SIMD
	Multiple	MISD	MIMD

SISD represents the organization containing single control unit, a processor unit and a memory unit. Instructions are executed sequentially and system may or may not have internal parallel processing capabilities.

SIMD represents an organization that includes many processing units under the supervision of a common control unit.

MISD structure is of only theoretical interest since no practical system has been constructed using this organization.

MIMD organization refers to a computer system capable of processing several programs at the same time.

The main difference between multicomputer system and multiprocessor system is that the multiprocessor system is controlled by one operating system that provides interaction between processors and all the component of the system cooperate in the solution of a problem.

Parallel Processing can be discussed under following topics:

- ✓ Pipeline Processing
- ✓ Vector Processing
- ✓ Array Processors

Pipelining:

A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.

It is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segments that operates concurrently with all other segments.

Each segment performs partial processing dictated by the way task ispartitioned.

The result obtained from each segment is transferred to next segment.

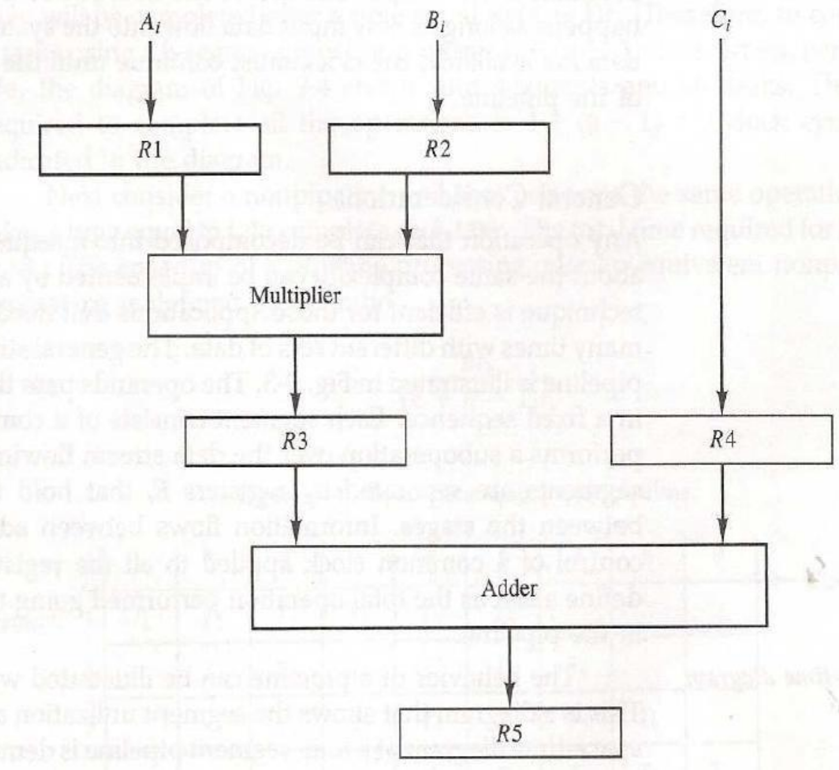
The final result is obtained when data have passed through all segments.

Suppose we have to perform the following task:

Each sub operation is to be performed in a segment within a pipeline. Each segment has one or two registers and a combinational circuit.

$$A_i * B_i + C_i \qquad \text{for } i = 1, 2, 3, \ldots, 7$$

Figure Example of pipeline processing.

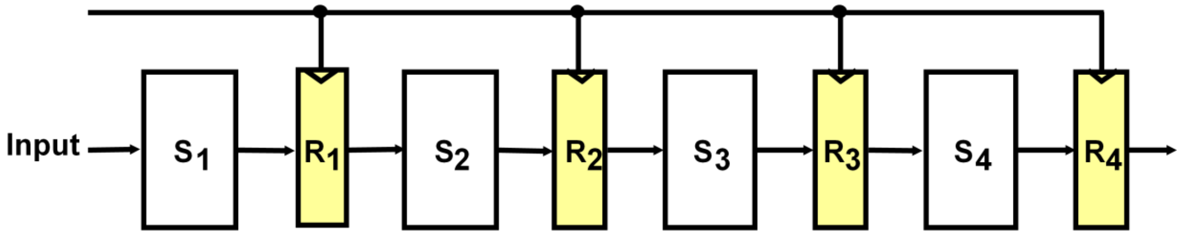


Suboperations in each segment:		
	$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Load A_i and B_i
	$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply and load C_i
	$R5 \leftarrow R3 + R4$	Add

Operations in each Pipeline Stage:

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	---	---	-------
2	A2	B2	A1 * B1	C1	-------
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

General Structure of a 4-Segment Pipeline



Space-Time Diagram

The following diagram shows 6 tasks T1 through T6 executed in 4segments.

Clock cycles

Segment	1	2	3	4	5	6	7	8	9
1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

No matter how many segments, once the pipeline is full, it takes only one clock period to obtain an output.

Pipeline Speed Up:

Consider the case where a k-segment pipeline used to execute n tasks.

$n = 6$ in previous example

$k = 4$ in previous example

Pipelined Machine (k stages, n tasks)

The first task t1 requires k clock cycles to complete its operation since thereare k segments

The remaining n-1 tasks require n-1 clock cycles

The n tasks clock cycles = $k+(n-1)$ (9 in previous example)

Conventional Machine (Non-Pipelined)

Cycles to complete each task in nonpipeline = k

For n tasks, n cycles required is Speedup (S)

S = Nonpipeline time /Pipeline time

For n tasks: $S = nk/(k+n-1)$

As n becomes much larger than k-1; Therefore, $S = nk/n = k$

Pipeline and Multiple Function Units:

Example:

4-stage pipeline

100 tasks to be executed

1 task in non-pipelined system; 4 clock cycles

Pipelined System : $k + n - 1 = 4 + 99 = 103$ clock cycles Non-Pipelined System : $n*k = 100 * 4 =$ 400 clock cyclesSpeedup :$S_k = 400 / 103 = 3.88$

Types of Pipelining:

- Arithmetic Pipeline
- Instruction Pipeline

Arithmetic Pipeline:

Pipeline arithmetic units are usually found in very high speed computers.

They are used to implement floating point operations.

We will now discuss the pipeline unit for the floating point addition and subtraction.

The inputs to floating point adder pipeline are two normalized floating point numbers.

A and B are mantissas and a and b are the exponents.

The floating point addition and subtraction can be performed in four segments. Floating-point adder:

- ✓ Compare the exponents
- ✓ Align the mantissa
- ✓ Add/sub the mantissa
- ✓ Normalize the result

$X = A \times 10^a = 0.9504 \times 10^3$

$Y = B \times 10^b = 0.8200 \times 10^2$

- ➢ Compare exponents : $3 - 2 = 1$
- ➢ Align mantissas

  $X = 0.9504 \times 10^3$

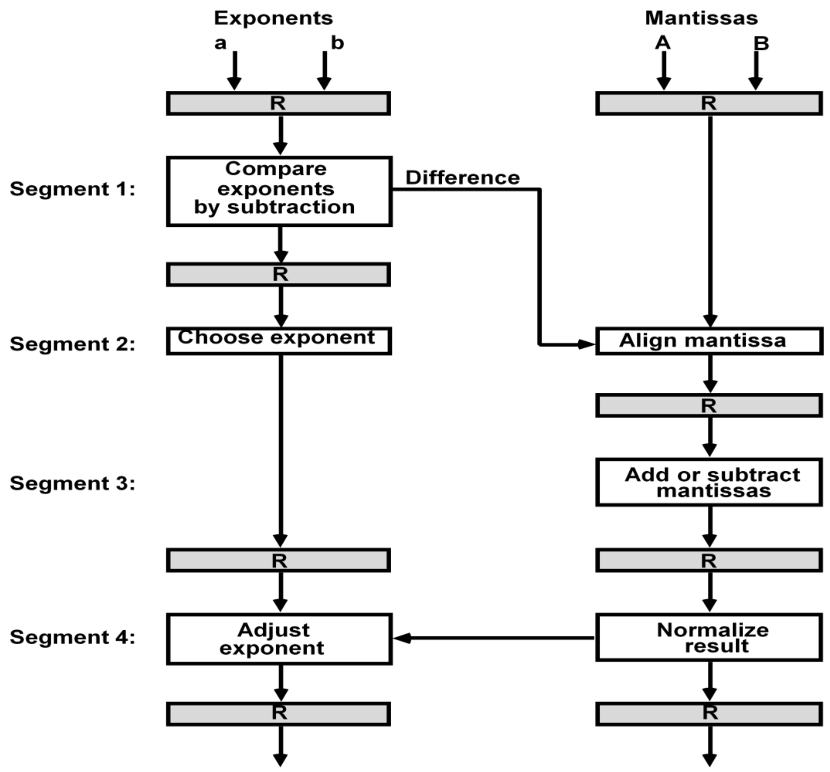
  $Y = 0.08200 \times 10^3$

- ➢ Add mantissas

  $Z = 1.0324 \times 10^3$

- ➢ Normalize result

  $Z = 0.10324 \times 10^4$



Instruction Pipeline:

Pipeline processing can occur not only in the data stream but in the instruction streamas well.

An instruction pipeline reads consecutive instruction from memory while previous instruction are being executed in other segments.

This caused the instruction fetch and execute segments to overlap and perform simultaneous operation.

Four Segment CPU Pipeline:

- FI segment fetches the instruction.
- DA segment decodes the instruction and calculate the effective address.
- FO segment fetches the operand.
- EX segment executes the instruction.

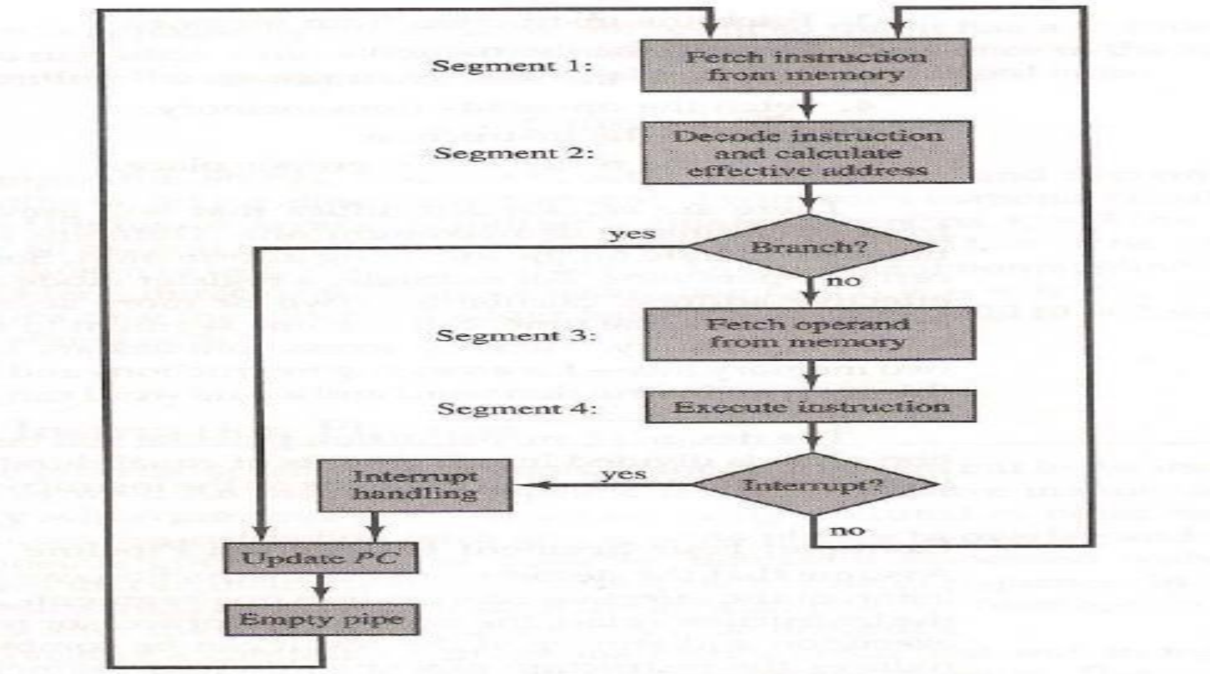


Figure Four-segment CPU pipeline.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Figure Timing of instruction pipeline.

Instruction Cycle:

Pipeline processing can occur also in the instruction stream. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. Six Phases* in an Instruction Cycle.

[1] Fetch an instruction from memory

[2] Decode the instruction

[3] Calculate the effective address of the operand

[4] Fetch the operands from memory

[5] Execute the operation

[6] Store the result in the proper place

Some instructions skip some phases

Effective address calculation can be done in the part of the decoding phase

Storage of the operation result into a register is done automatically in the execution phase

==> 4-Stage Pipeline

[1] FI: Fetch an instruction from memory

[2] DA: Decode the instruction and calculate the effective address of the operand

[3] FO: Fetch the operand

[4] EX: Execute the operation

Pipeline Conflicts :

Pipeline Conflicts : There are 3 major difficulties

In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. *Resource conflicts* caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
2. *Data dependency* conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. *Branch difficulties* arise from branch and other instructions that change the value of *PC*.

1) Resource conflicts: memory access by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2) Data dependency: when an instruction depend on the result of a previous instruction, but this result is not yet available.

   Example: an instruction with register indirect mode cannot proceed to fetch the operand if the previous instruction is loading the address into the register.

3) Branch difficulties: branch and other instruction (interrupt, ret, ..) that change the value of PC.

Handling Data Dependency:

This problem can be solved in the following ways:

Hardware interlocks: It is the circuit that detects the conflict situation and delayed the instruction by sufficient cycles to resolve the conflict.

Operand Forwarding: It uses the special hardware to detect the conflict and avoid it by routing the data through the special path between pipeline segments.

Delayed Loads: The compiler detects the data conflict and reorder the instruction as necessary to delay the loading of the conflicting data byinserting no operation instruction.

Handling of Branch Instruction:

- ✓ Pre fetch the target instruction.
- ✓ Branch target buffer(BTB) included in the fetch segment of the pipeline
- ✓ Branch Prediction

Delayed Branch RISC Pipeline:

Simplicity of instruction set is utilized to implement an instruction pipeline using small number of sub-operation, with each being executed in single clock cycle.

Since all operation are performed in the register, there is no need of effective addresscalculation.

Three Segment Instruction Pipeline:

I: Instruction Fetch

A: ALU Operation

E: Execute InstructionDelayed Load:

Consider now the operation of the following four instructions

1. LOAD: $R1 \leftarrow M[\text{address } 1]$
2. LOAD: $R2 \leftarrow M[\text{address } 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[\text{address } 3] \leftarrow R3$

Clock cycles:	1	2	3	4	5	6
1. Load $R1$	I	A	E			
2. Load $R2$		I	A	E		
3. Add $R1 + R2$			I	A	E	
4. Store $R3$				I	A	E

Pipeline timing with data conflict

Clock cycle:	1	2	3	4	5	6	7
1. Load $R1$	I	A	E				
2. Load $R2$		I	A	E			
3. No-operation			I	A	E		
4. Add $R1 + R2$				I	A	E	
5. Store $R3$					I	A	E

Pipeline timing with delayed load

Delayed Branch:

Let us consider the program having the following 5 instructions

Load from memory to $R1$
Increment $R2$
Add $R3$ to $R4$
Subtract $R5$ from $R6$
Branch to address X

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to X					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in X								I	A	E

Using no-operation instructions

Vector processing:

Vector processing is a central processing unit that can perform the complete vector input in individual instruction. It is a complete unit of hardware resources that implements a sequential set of similar data elements in the memory using individual instruction.

The scientific and research computations involve many computations which require extensive and high-power computers. These computations when run in a conventional computer may take days or weeks to complete. The science and engineering problems can be specified in methods of vectors and matrices using vector processing.

Features of Vector Processing

There are various features of Vector Processing which are as follows −

- A vector is a structured set of elements. The elements in a vector are scalar quantities. A vector operand includes an ordered set of n elements, where n is known as the length of the vector.
- Each clock period processes two successive pairs of elements. During one single clock period, the dual vector pipes and the dual sets of vector functional units allow the processing of two pairs of elements.

- As the completion of each pair of operations takes place, the results are delivered to appropriate elements of the result register. The operation continues just before the various elements processed are similar to the count particularized by the vector length register.
- In parallel vector processing, more than two results are generated per clock cycle. The parallel vector operations are automatically started under the following two circumstances −
    - ✓ When successive vector instructions facilitate different functional units and multiple vector registers.
    - ✓ When successive vector instructions use the resulting flow from one vector register as the operand of another operation utilizing a different functional unit. This phase is known as chaining.
- A vector processor implements better with higher vectors because of the foundation delay in a pipeline.
- Vector processing decrease the overhead related to maintenance of the loop-control variables which creates it more efficient than scalar processing.

Array processors:

Array processors are also known as multiprocessors or vector processors. They perform computations on large arrays of data. Thus, they are used to improve the performance of the computer.

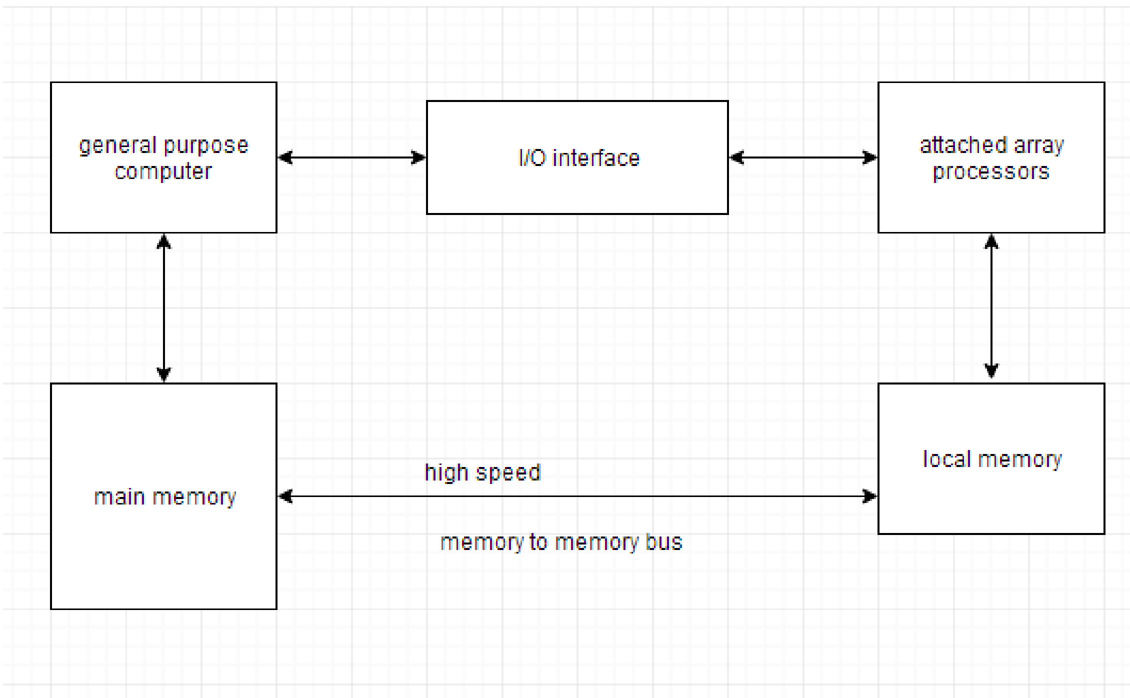
Types of Array Processors

There are basically two types of array processors:

- Attached Array Processors
- SIMD Array Processors

Attached Array Processors:

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.



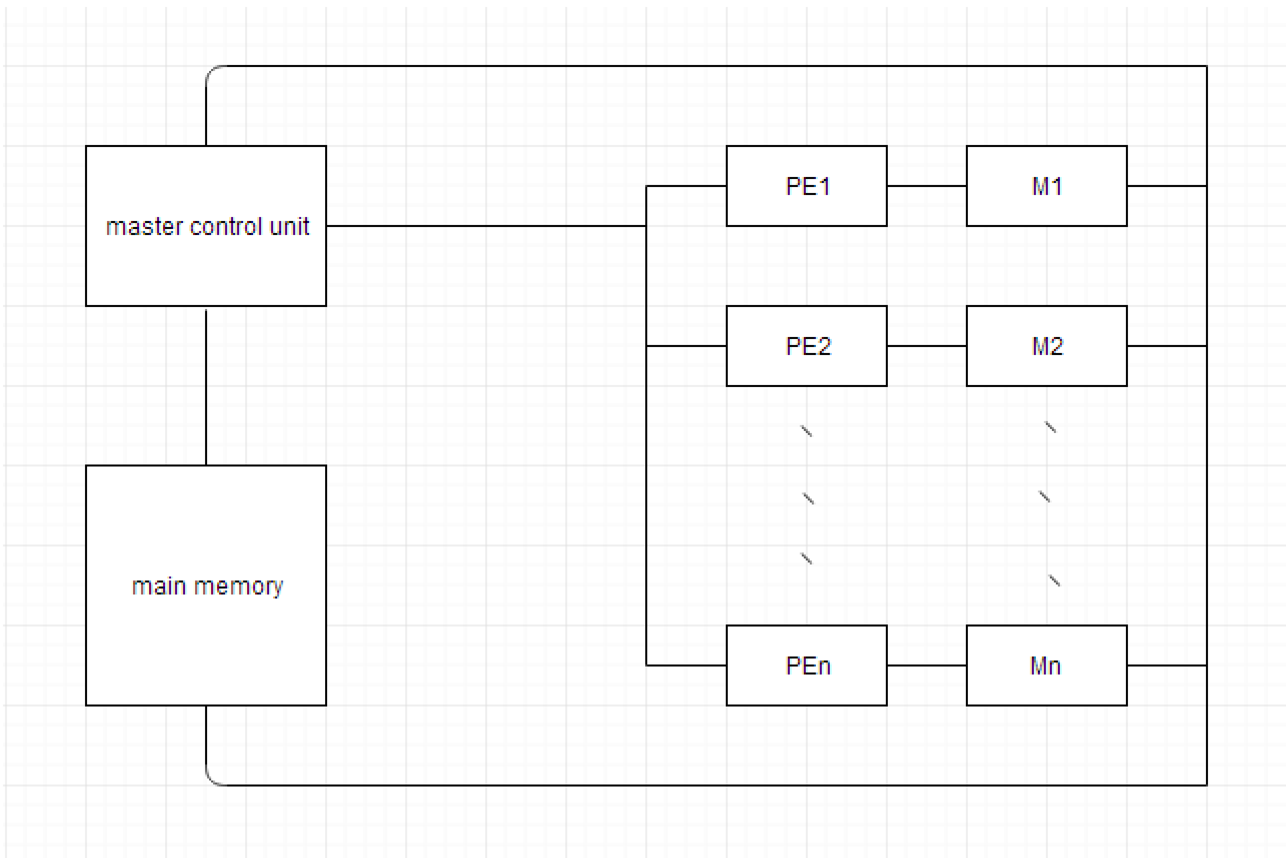
SIMD Array Processors

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an **ALU** and **registers**. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the **ILLIAC IV** computer developed by the **Burroughs corps**. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.



Why use the Array Processor:

- Array processors increases the overall instruction processing speed.
- As most of the Array processors operates asynchronously from the host CPU, hence it improves the overall capacity of the system.
- Array Processors has its own local memory, hence providing extra memory for systems with low memory.

UNIT – 05 – Part - C

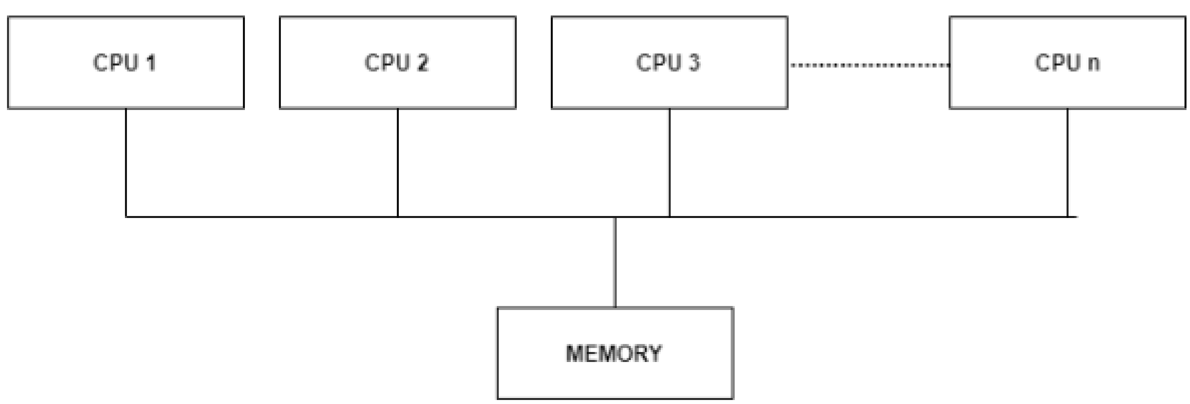
MULTIPROCESSORS

A multiprocessor is a computer system with two or more central processing units (CPUs), with each one sharing the common main memory as well as the peripherals. This helps in simultaneous processing of programs.

` The key objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.

A good illustration of a multiprocessor is a single central tower attached to two computer systems. A multiprocessor is regarded as a means to improve computing speeds, performance and cost-effectiveness, as well as to provide enhanced availability and reliability.

Most computer systems are single processor systems i.e they only have one processor. However, multiprocessor or parallel systems are increasing in importance nowadays. These systems have multiple processors working in parallel that share the computer clock, memory, bus, peripheral devices etc. An image demonstrating the multiprocessor architecture is −



Multiprocessing Architecture

Types of Multiprocessors:

There are mainly two types of multiprocessors i.e. symmetric and asymmetric multiprocessors. Details about them are as follows −

Symmetric Multiprocessors

In these types of systems, each processor contains a similar copy of the operating system and they all communicate with each other. All the processors are in a peer to peer relationship i.e. no master - slave relationship exists between them.

An example of the symmetric multiprocessing system is the Encore version of Unix for the Multimax Computer.

Asymmetric Multiprocessors

In asymmetric systems, each processor is given a predefined task. There is a master processor that gives instruction to all the other processors. Asymmetric multiprocessor system contains a master slave relationship.

Asymmetric multiprocessor was the only type of multiprocessor available before symmetric multiprocessors were created. Now also, this is the cheaper option.

Advantages of Multiprocessor Systems:

There are multiple advantages to multiprocessor systems. Some of these are −

More reliable Systems

In a multiprocessor system, even if one processor fails, the system will not halt. This ability to continue working despite hardware failure is known as graceful degradation. For example: If there are 5 processors in a multiprocessor system and one of them fails, then also 4 processors are still working. So the system only becomes slower and does not ground to a halt.

Enhanced Throughput

If multiple processors are working in tandem, then the throughput of the system increases i.e. number of processes getting executed per unit of time increase. If there are N processors then the throughput increases by an amount just under N.

More Economic Systems

Multiprocessor systems are cheaper than single processor systems in the long run because they share the data storage, peripheral devices, power supplies etc. If there are multiple processes that share data, it is better to schedule them on multiprocessor systems with shared data than have different computer systems with multiple copies of the data.

Disadvantages of Multiprocessor Systems:

There are some disadvantages as well to multiprocessor systems. Some of these are:

Increased Expense

Even though multiprocessor systems are cheaper in the long run than using multiple computer systems, still they are quite expensive. It is much cheaper to buy a simple single processor system than a multiprocessor system.

Complicated Operating System Required

There are multiple processors in a multiprocessor system that share peripherals, memory etc. So, it is much more complicated to schedule processes and impart resources to processes.than in single processor systems. Hence, a more complex and complicated operating system is required in multiprocessor systems.

Large Main Memory Required

All the processors in the multiprocessor system share the memory. So a much larger pool of memory is required as compared to single processor systems.

Characteristics of Multiprocessors:

A multiprocessor is a single computer that has multiple processors. It is possible that the processors in the multiprocessor system can communicate and cooperate at various levels of solving a given problem. The communications between the processors take place by sending messages from one processor to another, or by sharing a common memory.

There are the major characteristics of multiprocessors are as follows −

- **Parallel Computing** − This involves the simultaneous application of multiple processors. These processors are developed using a single architecture to execute a common task. In general, processors are identical and they work together in such a way that the users are under the impression that they are the only users of the system. In reality, however, many users are accessing the system at a given time.
- **Distributed Computing** − This involves the usage of a network of processors. Each processor in this network can be considered as a computer in its own right and have the capability to solve a problem. These processors are heterogeneous, and generally, one task is allocated to a single processor.
- **Supercomputing** − This involves the usage of the fastest machines to resolve big and computationally complex problems. In the past, supercomputing machines were vector computers but at present, vector or parallel computing is accepted by most people.
- **Pipelining** − This is a method wherein a specific task is divided into several subtasks that must be performed in a sequence. The functional units help in performing each subtask. The units are attached serially and all the units work simultaneously.
- **Vector Computing** − It involves the usage of vector processors, wherein operations such as ‘multiplication’ are divided into many steps and are then applied to a stream of operands (“vectors”).
- **Systolic** − This is similar to pipelining, but units are not arranged in a linear order. The steps in systolic are normally small and more in number and performed in a lockstep

manner. This is more frequently applied in special-purpose hardware such as image or signal processors.

Interconnection Structures:

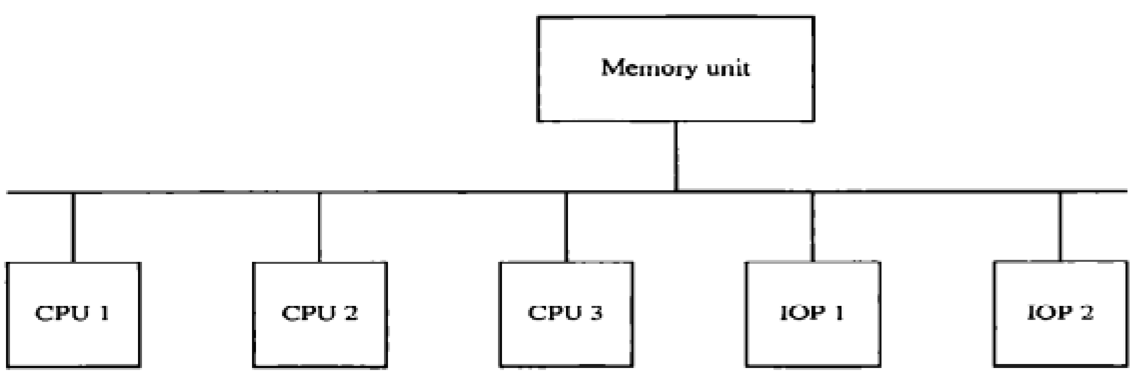
The interconnection between the components of a multiprocessor System can have different physical configurations depending n the number of transfer paths that are available between the processors and memory in a shared memory system and among the processing elements in a loosely coupled system.

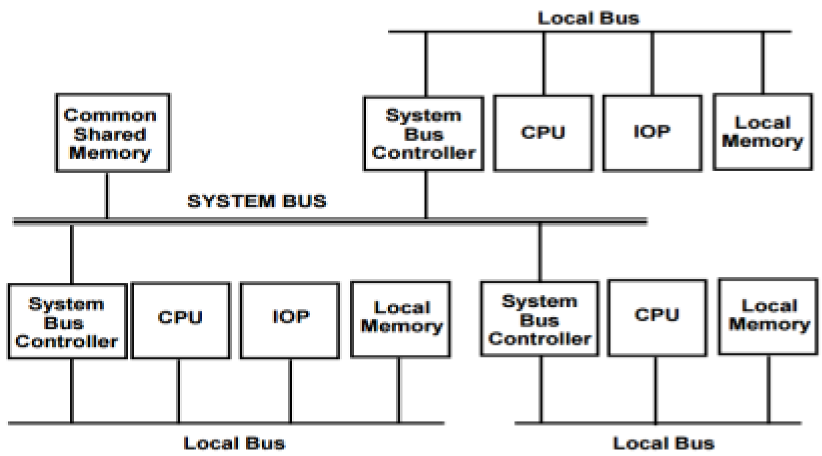
Some of the schemes are as: -

- Time-Shared Common Bus
- Multiport Memory
- Crossbar Switch
- Multistage Switching Network
- Hypercube System

Time shared common Bus:

- ✓ All processors (and memory) are connected to a common bus or busses
- ✓ Memory access is fairly uniform, but not very scalable
- ✓ A collection of signal lines that carry module-to-module communication
- ✓ Data highways connecting several digital system elements
- ✓ Operations of Bus





In the above figure we have number of local buses to its own local memory and to one or more processors. Each local bus may be connected to a CPU, an IOP, or any combinations of processors. A system bus controller links each local bus to a common system bus. The I/O devices connected to the local IOP, as well as the local memory, are available to the local processor. The memory connected to the common system bus is shared by all processors. If an IOP is connected directly to the system bus the I/O devices attached to it may be made available to all processors.

Disadvantage.:

- Only one processor can communicate with the memory or another processor at any given time.
- As a consequence, the total overall transfer rate within the system is limited by the speed of the single path.

Multiport Memory:

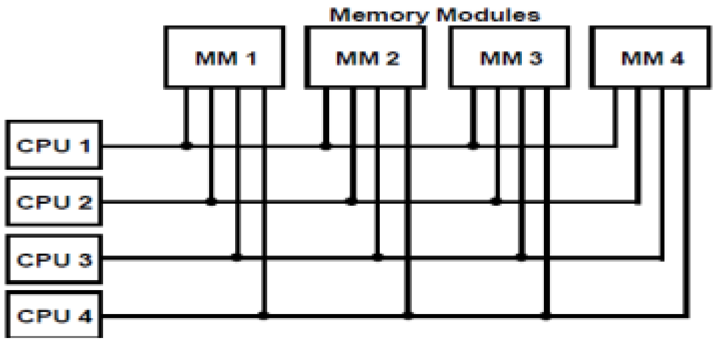
- ✓ Each port serves a CPU Memory Module Control Logic
- ✓ Each memory module has control logic
- ✓ Resolve memory module conflicts Fixed priority among CPUs

Advantages

The high transfer rate can be achieved because of the multiple paths.

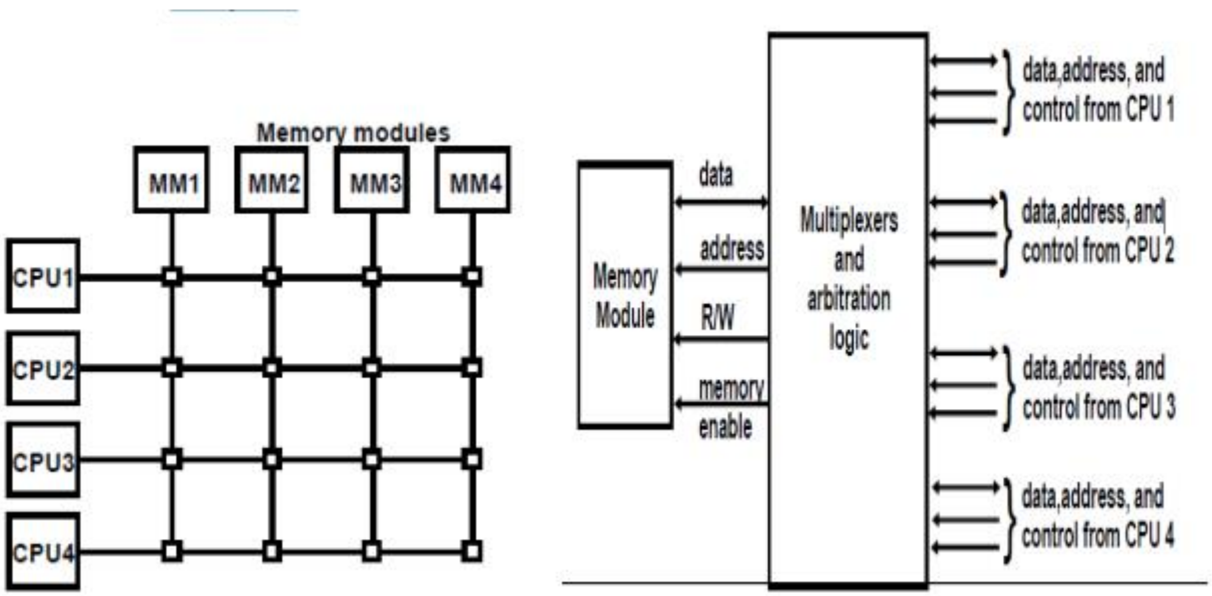
Disadvantages:

It requires expensive memory control logic and a large number of cables and connections



Crossbar switch:

- Each switch point has control logic to set up the transfer path between a processor and a memory.
- It also resolves the multiple requests for access to the same memory on the predetermined priority basis.
- Though this organization supports simultaneous transfers from all memory modules because there is a separate path associated with each Module.
- The H/w required to implement the switch can become quite large and complex



a) b)

Advantage:

Supports simultaneous transfers from all memory modules

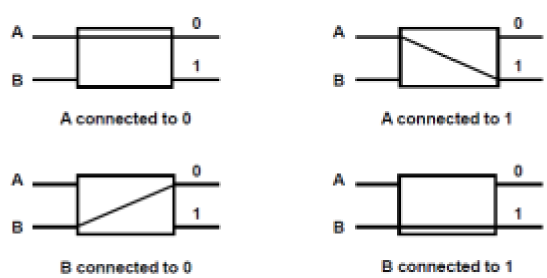
Disadvantage:

The hardware required to implement the switch can become quite large and complex.

Multistage Switching Network:

The basic component of a multi stage switching network is a two-input, twooutput interchange switch.

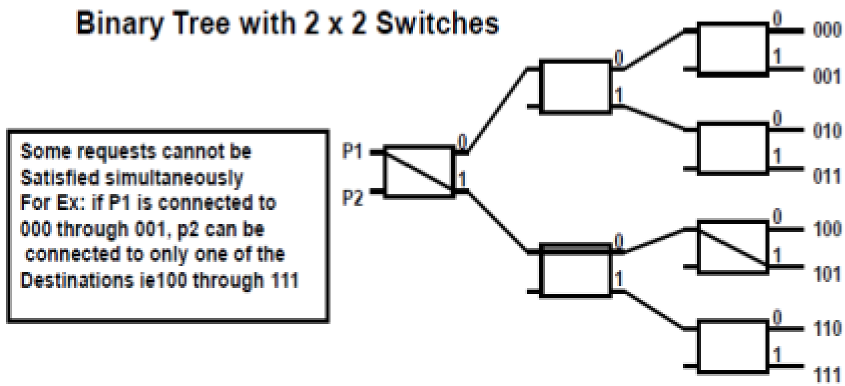
Interstage Switch

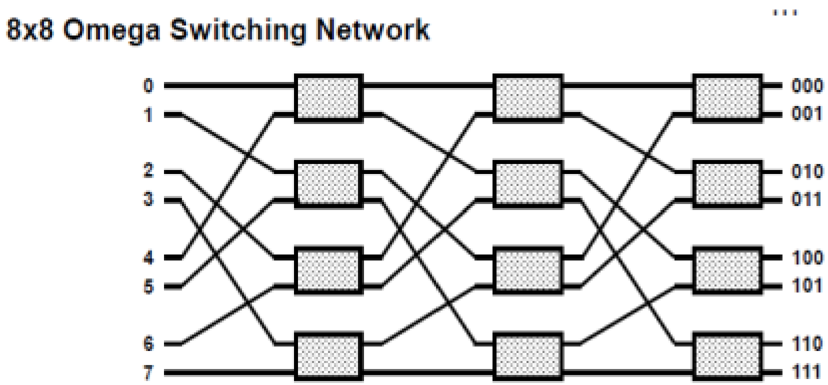


Using the 2x2 switch as a building block, it is possible to build a multistage network to control the communication between a number of sources and destinations.

To see how this is done, consider the binary tree shown in Fig. below.

Certain request patterns cannot be satisfied simultaneously. i.e., if P1 000~011, then P2 100~111





Some request patterns cannot be connected simultaneously. i.e., any two sources cannot be connected simultaneously to destination 000 and 001

In a tightly coupled multiprocessor system, the source is a processor and the destination is a memory module.

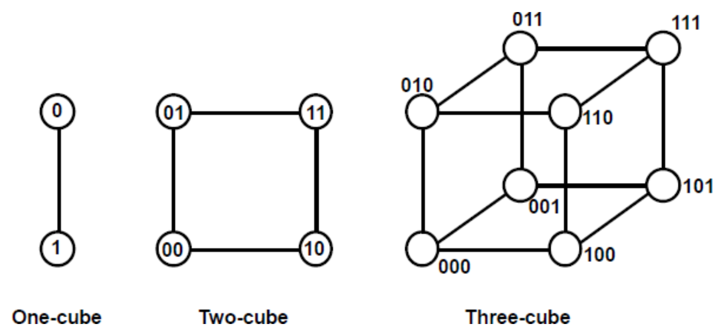
Set up the path transfer the address into memory transfer the data

In a loosely coupled multiprocessor system, both the source and destination are Processsing elements.

Hypercube System:

The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of N=2n processors interconnected in an n-dimensional binary cube.

- Each processor forms a node of the cube, in effect it contains not only a CPUbut also local memory and I/O interface.
- Each processor address differs from that of each of its n neighbors by exactlyone bit position.
- Fig. below shows the hypercube structure for n=1, 2, and 3.
- Routing messages through an *n*-cube structure may take from one to *n* links from a source node to a destination node.
- A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address.
- The message is then sent along any one of the axes that the resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ.
- A representative of the hypercube architecture is the Intel iPSC computer complex.
- It consists of 128(*n*=7) microcomputers, each node consists of a CPU, a floating point processor, local memory, and serial communication interface units



Hypercube structures for n=1,2,3

Inter-processor Arbitration:

Only one of CPU, IOP, and Memory can be granted to use the bus at a time

Arbitration mechanism is needed to handle multiple requests to the shared resources to resolve multiple contention

System Bus:

- ✓ A bus that connects the major components such as CPU’s, IOP’s and memory
- ✓ A typical System bus consists of 100 signal lines divided into three functional groups: data, address and control lines. In addition there are power distribution lines to the components.

Synchronous Bus:

- ✓ Each data item is transferred over a time slice
- ✓ known to both source and destination unit
- ✓ Common clock source or separate clock and synchronization signal is transmitted periodically to synchronize the clocks in the system

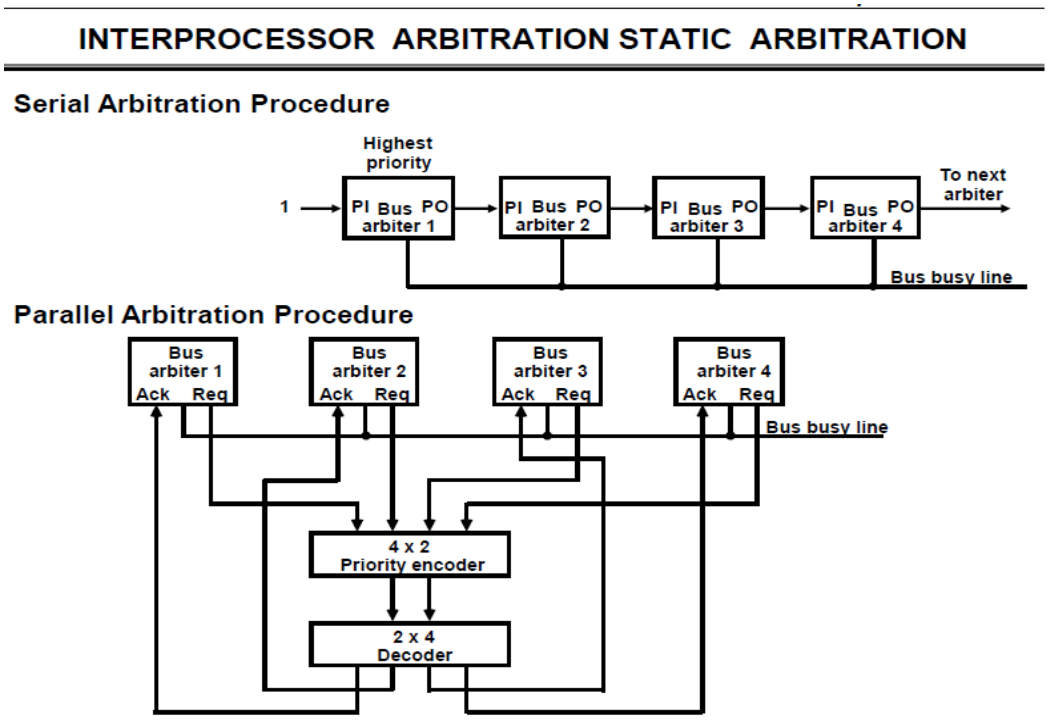
Asynchronous Bus:

- ✓ Each data item is transferred by Handshake mechanism
  - ➢ Unit that transmits the data transmits a control signal that indicates the presence of data
  - ➢ Unit that receiving the data responds with another control signal to acknowledge the receipt of the data
- ✓ Strobe pulse -supplied by one of the units to indicate to the other unit

when the data transfer has to occur

Table IEEE standard 796 multibus signals

	Signal name
Data and address	
Data lines (16 lines)	DATA0–DATA15
Address lines (24 lines)	ADRS0–ADRS23
Data transfer	
Memory read	MRDC
Memory write	MWTC
IO read	IORC
IO write	IOWC
Transfer acknowledge	TACK
Interrupt control	
Interrupt request (8 lines)	INT0–INT7
Interrupt acknowledge	INTA
Miscellaneous control	
Master clock	CCLK
System initialization	INIT
Byte high enable	BHEN
Memory inhibit (2 lines)	INH1–INH2
Bus lock	LOCK
Bus arbitration	
Bus request	BREQ
Common bus request	CBRQ
Bus busy	BUSY
Bus clock	BCLK
Bus priority in	BPRN
Bus priority out	BPRO
Power and ground (20 lines)	



Inter-processor arbitration static arbitration

Interprocessor Arbitration Dynamic Arbitration:

Priorities of the units can be dynamically changeable while the system is in operation

Time Slice

Fixed length time slice is given sequentially to each processor, round- robin fashion

Polling

Unit address polling -Bus controller advances the address to identify the requesting unit. When processor that requires the access recognizes its address, it activates the bus busy line and then accesses the bus. After a number of bus cycles, the polling continues by choosing a different processor.

LRU

The least recently used algorithm gives the highest priority to the requesting device that has not used bus for the longest interval.

FIFO

The first come first serve scheme requests are served in the order received. The bus controller here maintains a queue data structure.

Rotating Daisy Chain

Conventional Daisy Chain -Highest priority to the nearest unit to the bus controller

Rotating Daisy Chain –The PO output of the last device is connected tothe PI of the first one. Highest priority to the unit that is nearest to the unit that has most recently accessed the bus(it becomes the bus controller)

Inter processor communication and synchronization:

The various processors in a multiprocessor system must be provided with a facility for *communicating* with each other.

A communication path can be established through *a portion of memory* or *a common input-output channels*.

The sending processor structures a request, a message, or a procedure, andplaces it in the memory mailbox.

- ✓ *Status bits* residing in common memory
- ✓ The receiving processor can check the mailbox *periodically*.
- ✓ The response time of this procedure can be time consuming.

A more efficient procedure is for the sending processor to alert the receivingprocessor directly by means of an *interrupt signal*.

In addition to shared memory, a multiprocessor system may have other sharedresources.

e.g., a magnetic disk storage unit.

- ❖ To prevent conflicting use of shared resources by several processors there must be a provision for assigning resources to processors. i.e., operating system.
- ❖ There are three organizations that have been used in the design of operating system for multiprocessors: *master-slave configuration*, *separate operating system*, and *distributed operating system*.
- ❖ In a master-slave mode, one processor, master, always executes the operating system functions.
- ❖ In the separate operating system organization, each processor can execute the operating

system routines it needs. This organization is more suitable for *loosely coupled systems*.

- ❖ In the distributed operating system organization, the operating system routines are distributed among the available processors. However, each particular operating system function is assigned to only one processor at a time. It is also referred to as a *floating operating system*.

Loosely Coupled System:

- There is *no shared memory* for passing information.
- The communication between processors is by means of message passing through *I/O channels*.
- The communication is initiated by one processor calling a *procedure* that resides in the memory of the processor with which it wishes to communicate.
- The communication efficiency of the interprocessor network depends on the *communication routing protocol, processor speed, data link speed, and the topology of the network*.

Interprocess Synchronization:

The instruction set of a multiprocessor contains basic instructions that are used to implement communication and synchronization between cooperating processes.

- Communication refers to the exchange of data between different processes.
- Synchronization refers to the special case where the data used to communicate between processors is control information.

Synchronization is needed to enforce the *correct sequence of processes* and to ensure *mutually exclusive access* to shared writable data.

Multiprocessor systems usually include various mechanisms to deal with the synchronization of resources.

- ❖ Low-level primitives are implemented directly by the hardware.
- ❖ These primitives are the basic mechanisms that enforce mutual exclusion for more complex mechanisms implemented in software.
- ❖ A number of hardware mechanisms for mutual exclusion have been developed.

Mutual Exclusion with Semaphore:

A properly functioning multiprocessor system must provide a mechanism that will guarantee orderly access to shared memory and other shared resources.

- ❖ Mutual exclusion: This is necessary to protect data from being changed simultaneously by two or more processors.
- ❖ Critical section: is a program sequence that must complete executionbefore another processor accesses the same shared resource.

A *binary variable* called a *semaphore* is often used to indicate whether or not aprocessor is executing a critical section.

Testing and setting the semaphore is itself a critical operation and must be performed as a single indivisible operation.

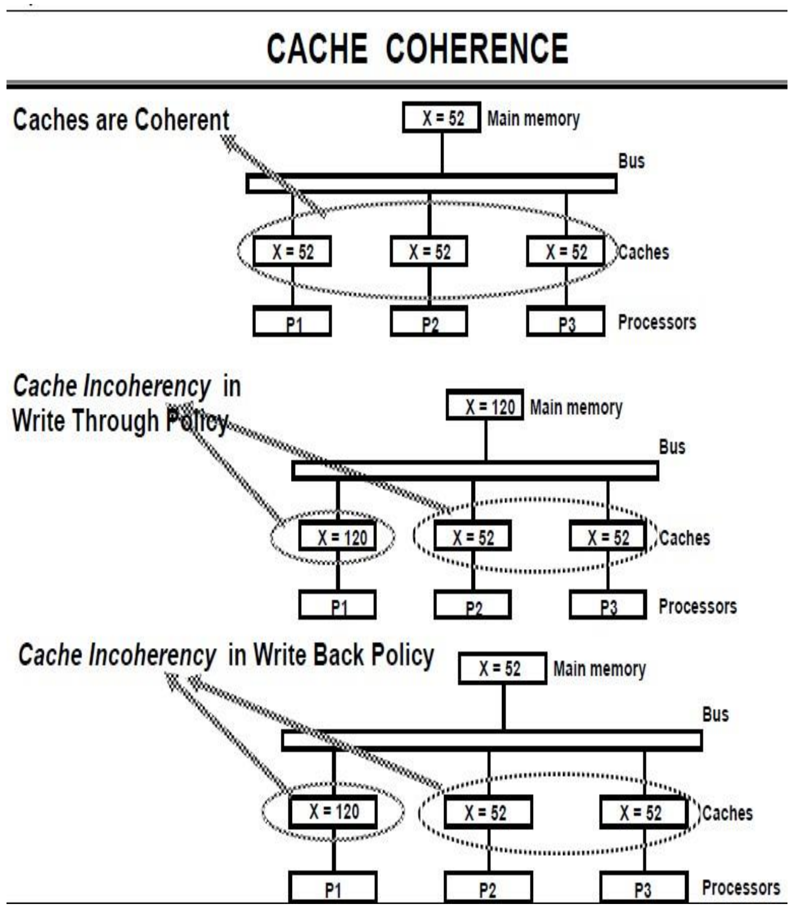
A semaphore can be initialized by means of a *test and set instruction* in conjunction with a hardware *lock* mechanism.

The instruction TSL SEM will be executed in two memory cycles (the first to read and the second to write) as follows:

$$R \leftarrow M[SEM],\ M[SEM] \leftarrow 1$$

Cache Coherence:

Cache coherence is the consistency of shared resource data that ends up stored in multiple local **caches**. When clients in a system maintain **caches** of a common memory resource, problems may arise with inconsistent data, which is particularly the case with CPUs in a multiprocessing system.



Shared Cache

- ✓ Disallow private cache
- ✓ Access time delaySoftware Approaches

Read-Only Data are Cacheable

- ✓ Private Cache is for Read-Only data
- ✓ Shared Writable Data are not cacheable
- ✓ Compiler tags data as cacheable and noncacheable
- ✓ Degrade performance due to software overhead

Centralized Global Table

- ✓ Status of each memory block is maintained in CGT: RO(Read-Only); RW(Read and Write)
- ✓ All caches can have copies of RO blocks

- ✓ Only one cache can have a copy of RW block
- ✓ Hardware Approaches

Snoopy Cache Controller

- ✓ Cache Controllers monitor all the bus requests from CPUs and IOPs
- ✓ All caches attached to the bus monitor the write operations
- ✓ When a word in a cache is written, memory is also updated (write through)
- ✓ Local snoopy controllers in all other caches check their memory to determine if they have a copy of that word; If they have, that location is marked invalid(future reference to this location causes cache miss)