GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356

Department of Computer Science and Engineering

HANDOUT

on

COMPUTER ORGANIZATION AND ARCHITECTURE

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth

Program Educational Objectives

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations
- Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON COMPUTER ORGANIZATION AND ARCHITECTURE

Class & Sem. : II B.Tech – II Semester Year: 2019-20

Branch : CSE Credits: 3

==========

1. Brief History and Scope of the Subject

The term *Computer Architecture* was first defined in the paper by Amdahl, Blaauw and Brooks of IBM Corporation announcing IBM System/360 computer family on April 7,1964. On that day IBM Corporation introduced, in the words of IBM spokesman, *"the most important product announcement that this corporation has made in its history"*.There were six models introduced originally, ranging in performance from 25 to 1. Six years later this performance range was increased to about 200 to 1.

This was the keyfeature which prompted IBM's effort to design architecture for a new line of computers that are to be code compatible with each other. The recognition that *architecture* andimplementation could be separated and that one need not imply the other led to establishment of a common System/360 machine architecture implemented in the range of models.

Recent Developments

- machine level representation of data
- assembly level machine organization
- memory system organization and architecture
- interfacing and communication

2. Pre-Requisites

- Binary arithmetic operations
- Operations of MUX, DEMUX, ENCODER, DECODER and Registers.

3. Course Objectives

- To familiarize with organizational aspects of memory, processor and I/O

4. Course Outcomes

Upon successful completion of the course, the studentswill be able to

CO1: identify different types of instructions.

CO2:differentiate micro-programmed and hard-wired control units.

CO3:analyze the performance of the hierarchical organization of memory.

CO4:demonstrate various operations on fixed and floating point numbers.

CO5:summarize different data transfer techniques.

CO6:demonstrate the use of parallel processing.

5. Program Outcomes and Program Specific Outcomes

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Students will be able to

- Design, develop, test and maintain reliable software systems and intelligent systems.
- Design and develop web sites, web apps and mobile apps.

6. Mapping of Course Outcomes with Program Outcomes and Program Specific Outcomes:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3	3	3											
CO2	3	3	3										2	
CO3	3	2	2										2	
CO4	3	3	3									2	2	
CO5	3	2	2									2	2	
CO6	3	2	2									2	2	

7. Prescribed Text Books

1. M.Moris Mano, Computer Systems Architecture, Pearson/PHI, 3rd edition

8. Reference Text Books

1. Carl Hamacher, Zvonks Vranesic, SafeaZaky, Computer Organization, McGraw Hill, 5th edition.
2. William Stallings, Computer Organization and Architecture, Pearson/PHI, 6th edition.
3. John L. Hennessy and David A. Patterson, Computer Architecture a quantitative approach, Elsevier, 4th Edition.

9. URLs and Other E-Learning Resources

Journals

- History of computing
- Computational science & Engineering
- Computer & Digital techniques.

URL's

1. https://www.geeksforgeeks.org/computer-organization-and-architecture-tutorials/

10. Digital Learning Materials:

1. SONET volumes -8
2. Computer Architecture --38 volumes

   By Prof.Anshul Kumar

   Dept.of Comp.sc.&Engg

   I.I.T. Delhi.

3. Computer Organization –33 volumes

By Prof. S.RAMAN

Dept.of Comp.sc.&Engg

I.I.T. MADRAS.

11. Lecture Schedule / Lesson Plan

Topic	No. of Periods	
	Theory	Tutorial
UNIT –1: Register transfer language and Micro operations		
Functional Units	1	1
Computer Registers, Register Transfer Languages	1	
Register Transfer, Bus and Memory Transfers	1	
Arithmetic Micro operations	2	
Logic Micro operations,	1	
Shift Micro operations, Arithmetic logic shift unit.	2	1
Instruction codes	1	
Instruction Cycle	1	
Register reference instructions	1	
Memory– Reference Instructions	1	1
Input – Output and Interrupt	2	
Total	**14**	**3**
UNIT – 2: CPU and Micro Programmed Control		
Instruction Formats	1	1
Addressing Modes	1	
Control Memory	1	1
Address Sequencing	1	
Design of Control Unit- Hard Wired Control, Micro Programmed Control	2	
Total	**6**	**2**
UNIT – 3: Memory Organization		
Memory Hierarchy	1	1
Main Memory	1	
Auxiliary Memory	1	
Associative Memory	1	1
Cache Memory	2	
Virtual Memory	1	
Total	**7**	**2**
UNIT – 4: Computer Arithmetic		
Addition and Subtraction – Fixed Point	1	1
Multiplication Algorithms– Fixed Point	2	
Division Algorithms– Fixed Point	2	
Floating – Point Arithmetic Operations	2	1
Total	**7**	**2**
UNIT – 5: Input-Output Organization		
Peripheral Devices, Input-Output Interface	1	1
Asynchronous data transfer	1	
Modes of Transfer, Priority Interrupt	2	
Direct memory Access	2	1
Input –Output Processor (IOP)	1	
Total	**7**	**2**
UNIT – 6: Parallel Processing		

Parallel Processing	1	1
Pipelining	1	
Arithmetic Pipeline, Instruction Pipeline	3	
Multi Processors: Characteristics of Multiprocessors	1	
Interconnection Structures	1	1
Inter Processor Arbitration	1	
Cache Coherence.	1	
Total	**9**	**2**
Total No.of Periods:	**50**	**13**

12. Seminar Topics:

Booths multiplication algorithm

Inter Processor Arbitration

UNIT – I

Register transfer language and Micro operations

Objective:

- To familiarize with organizational aspects of memory, processor and I/O.

Syllabus:

Functional units, Computer Registres, Register Transfer language, Register Transfer Bus and memory transfers, Arithmetic, logic and shift micro-operations, Arithmetic logic shift unit.
Instruction codes, Instruction cycle, register reference instructions, Memory – reference instructions, Input – Output and Interrupt.

Learning Outcomes:

At the end of the unit student will be able to:

1. Understand different types of instructions.
2. Describe about instruction cycle.

Learning Material

1.1 FUNCTIONAL UNIT

- A computer in its simplest form comprises of five functional units namely input unit, output unit, memory unit, arithmetic & logic unit and control unit. Below figure depicts the functional units of a computer system.

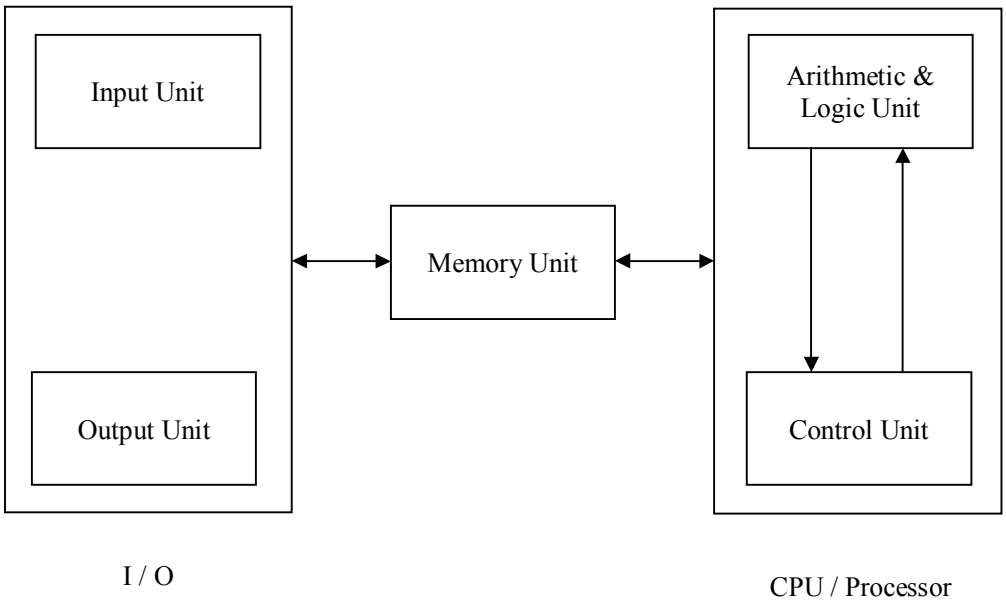


Figure 1: Basic functional units of a computer

1. Input Unit: Computer accepts encoded information through input unit. The standard input device is a keyboard. Whenever a key is pressed, keyboard controller sends the code to CPU/Memory.
Examples include Mouse, Joystick, Tracker ball, Light pen, Digitizer, Scanner etc.
2. Output Unit: Computer after computation returns the computed results, error messages, etc. via output unit. The standard output device is a video monitor, LCD/TFT monitor. Other output devices are printers, plotters etc.
3. Memory Unit: Memory unit stores the program instructions (Code), data and results of computations etc.
Memory unit is classified as:

- Primary /Main Memory
- Secondary /Auxiliary Memory

4. Arithmetic and logic unit: ALU consist of necessary logic circuits like adder, comparator etc., to perform operations of addition, multiplication, comparison of two numbers etc.
5. Control Unit: Control unit co-ordinates activities of all units by issuing control signals. Control signals issued by control unit govern the data transfers and then appropriate operations take place.
Control unit interprets or decides the operation/action to be performed.

1.2 COMPUTER REGISTERS

- Computer registers are designated by capital letters (sometimes followed by numerals) 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.
- The following diagram shows the representation of registers.

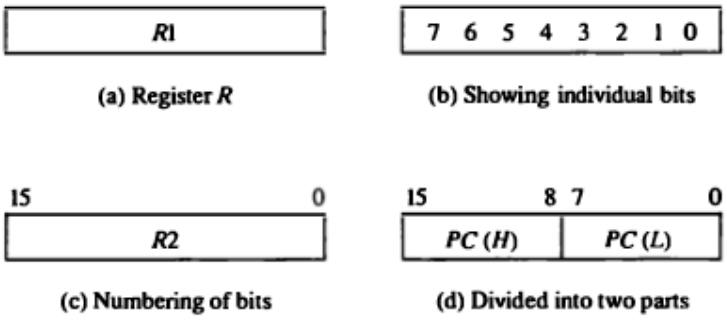


Figure 2: Block diagram of register

Table 1: 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

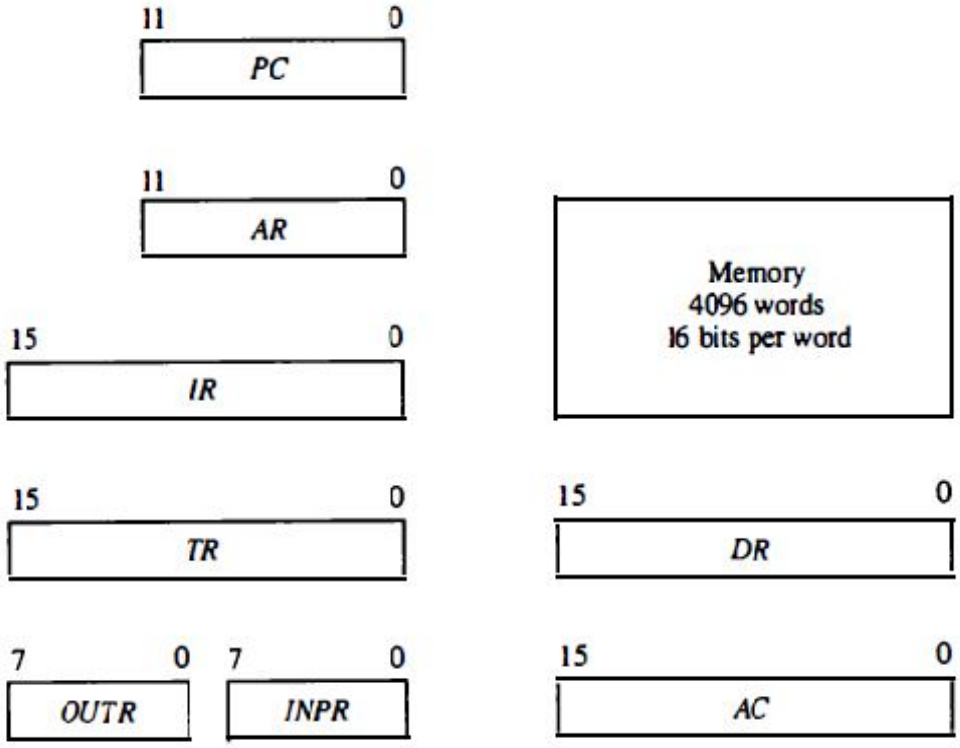


Figure 3: Basic computer registers and memory

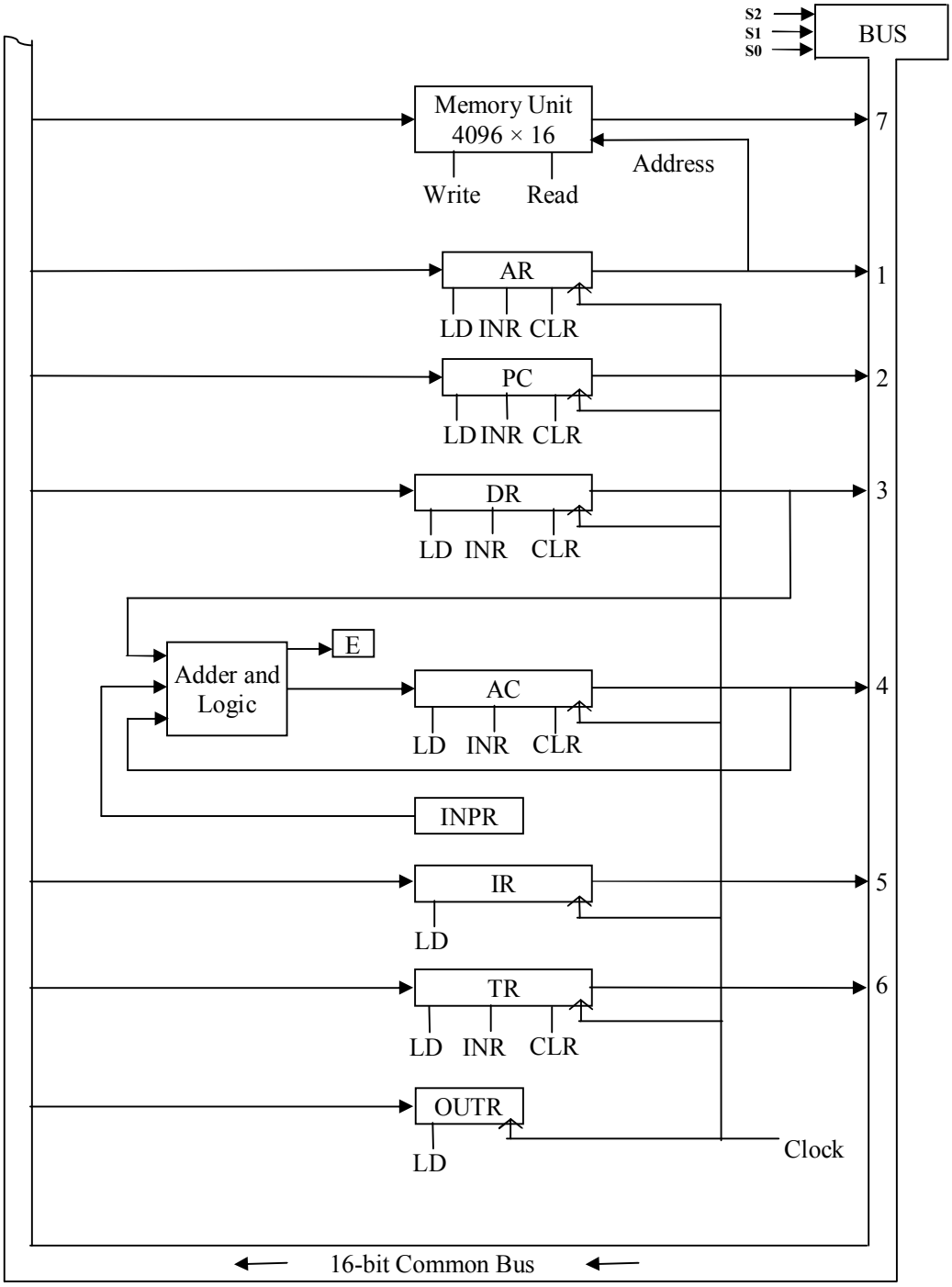


Figure 4: Basic Computer Register Connected to a Common BUS

- The outputs of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.

- The number along each output shows the decimal equivalent of the required binary selection. 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 since this is the binary value of decimal 3. The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse. The memory receives the contents of the bus when its write pin is enabled.
- The memory places its 16-bit output onto the bus when the read pin is enabled and $S_2S_1S_0$=11.
- Four registers, D R, AC, IR, and TR, have 16 bits each. Two registers, AR and PC, have 12 bits each since they hold a memory address.
- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to 0's. When AR or PC receives information from the bus, only the 12 least significant bits are transferred into the register.
- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus.
- INPR receives a character from an input device to provide information onto the bus which in turn is then transferred to AC.
- OUTR receives a character from AC and delivers it to an output device. There is no transfer from OUTR to any of the other registers.
- 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 synchronous clear.

Micro operation: is an elementary operation performed on information stored in one or more registers.

Eg: Shift, Count, Clear and Load.

1.3 REGISTER TRANSFER LANGUAGE

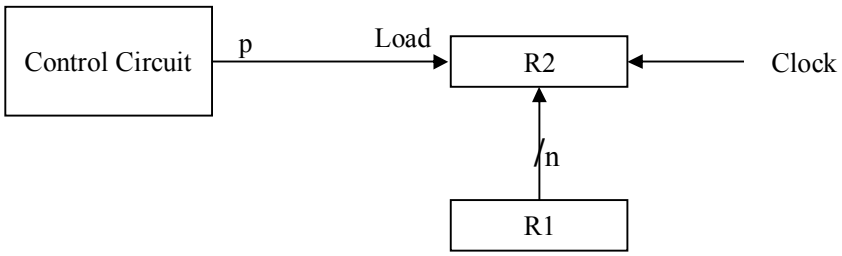
- The symbolic notation used to describe the Micro operation transfers among registers is called a Register Transfer Language.
- Information transferred from one register to another register is designed in symbolic form by means of replacement operator(←).
- The statement R2 ← R1 denotes a transfer of the contents of register R1 into register R2.
- If we want to transfer only under a predefined condition, this can be shown by means of if-then statement.

  if(p = 1) then R2 ← R1

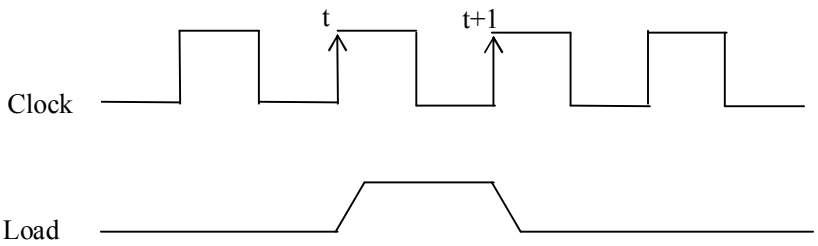
where p is a control signal generated in the control section.

- A Control function is a Boolean variable that is equal to 0 or 1.
- The control function included in the statement is represented as follows.

  p: R2 ← R1
- Here the transfer operation is performed by hardware only if p = 1.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.



(a) Block Diagram



(b) Timing Diagram

Figure 5: Transfer from R1 to R2 when p =1

The basic symbols of the register transfer notation are listed in the following table.

Table 2: Basic Symbols for Register Transfer

S.No	Symbol	Description	Example
1	Letters (and Numbers)	Denotes a register	MAR, R2
2	Parenthesis ( )	Denotes a part of register	R2(0-7), R2(L)
3	Arrow ←	Denotes transfer of information	R2 ← R1
4	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.

Eg: R2 ← R1, R1 ← R2

- The above statement denotes an operation that exchanges the content of two registers during one common clock pulse.

1.3.1 Bus Transfer

- Digital computers have many registers, and paths must be provided to transfer information from one register to another register.
- The no. of wires will be excessive if separate lines are used between each registers and all other registers in the system.
- A Bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
- Control signal determines which register is selected by the bus during each particular register transfer.
- The construction of a bus system for 4 registers is shown in the following figure.

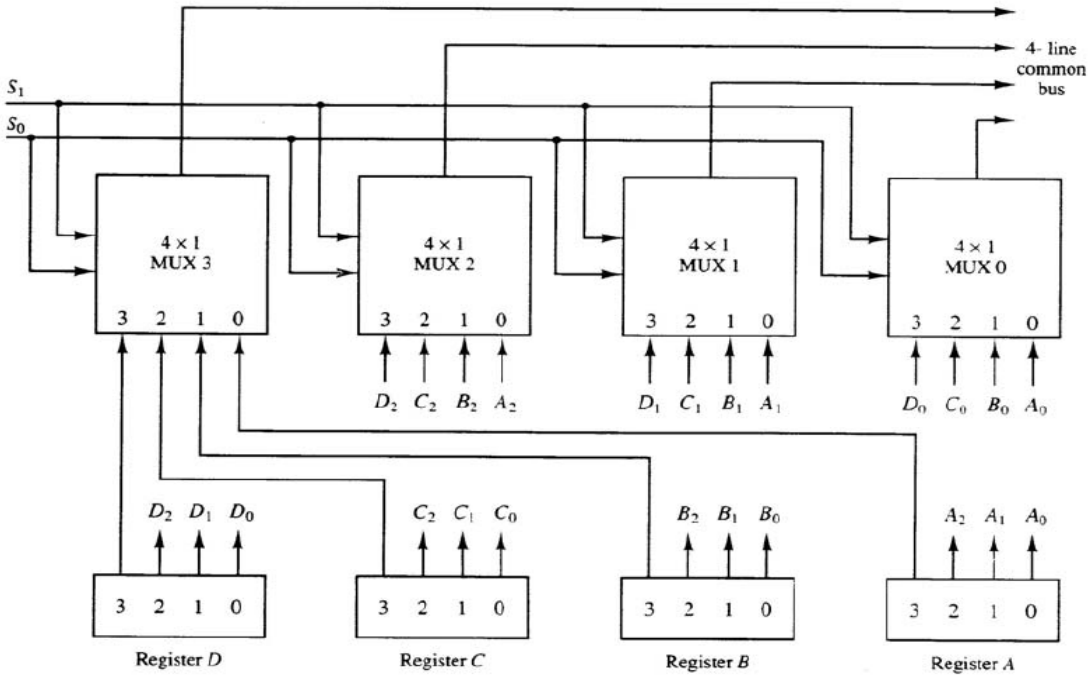


Figure 6: Bus system for four registers

- Here each register has 4 bits, numbered 0 through 3.
- The bus consists of four 4×1 multiplexers, each having four data inputs, 0 through 3, and two selection inputs S_1 and S_0.
- The selection lines choose the four bits of one register and transfer them into the 4-line common bus.

- When S_1S_0 = 00, then 0 data input of all four multiplexers are selected and applied to output that form the bus.
- The following table shows the register that is selected by the bus for each of the four possible binary values of selection lines.

Table 3: Selection of Registers

S_1	S_2	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

- The transfer of information from a bus into one of many registers can be accomplished by connecting the bus lines to inputs of all destination registers and activating the load control of particular destination register.

  Eg: BUS ← C
- In the above statement the content of register C is placed onto BUS

  Eg: R1 ← BUS
- In the above statement the content of register C, placed onto BUS is loaded into Register R1by activating the load pin.

1.3.1.1 Three (Tri) state Bus Buffer

- A Bus system can be constructed with three-state gates instead of multiplexers.
- A three-state gate is a digital circuit that exhibits three states.
- Two of the states are signals equivalent to logic 1 or 0 as a conventional gate.
- The third state is a high-impedance state. The high-impedance behaves like an open circuit.
- The graphical symbol of a three state buffer is shown in the following figure.

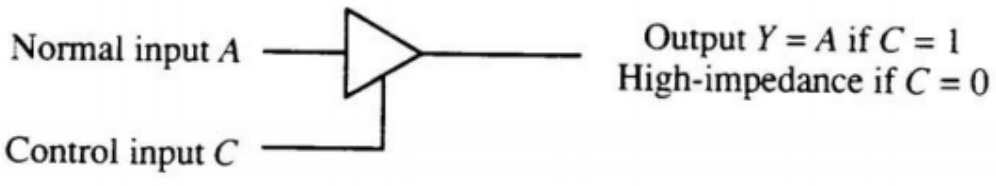


Figure 7: Graphic symbol of a three state buffer

- In the above diagram, control input determines the output state.
- When the control input is equal to 1, the output is enabled like as any conventional buffer.

- When the control input is equal to 0, the output is disabled and the gate goes to a high-impedance state.
- The following figure shows construction of a bus system with Three state Bus Buffer.

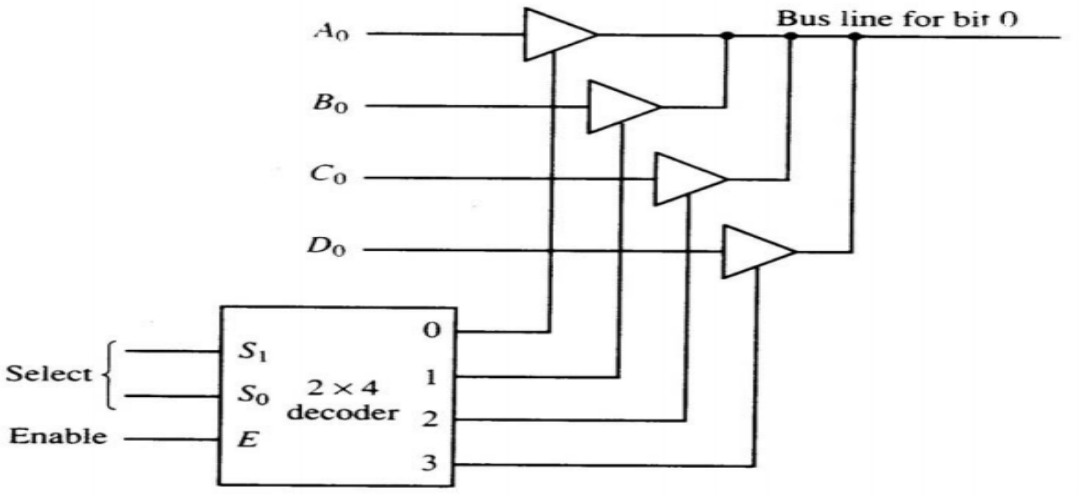


Figure 8: Bus Line with three state buffer

- Here the output of 4 buffers are connected together to form a single bus line.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- No more than one buffer may be in the active state at any given time.
- Only one three-state buffer has access to the bus line while all other buffers are maintained in high impedance state.
- With the help of the decoder, three state buffers are controlled.
- When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, i.e. 1, one of the three-state buffers will be active, depending on the select lines of the decoder.
- To construct a common bus for four registers of n bits each using three-state buffers, we need n circuits with four buffers in each circuit.
- Each group of four buffers receives one significant bit from the four registers.
- Each common output produces one of the lines for the common bus for a total of n lines.

Table 4: Active Three State Buffer with respect to enable input

E	S_1 S_0	Active Three State Buffer
0	X X	High impedance state
1	0 0	A

1	0	1	B
1	1	0	C
1	1	1	D

1.3.2 Memory Transfer

- The transfer of information from a memory word symbolized by M, to the outside environment is called a *read* operation.
- The transfer of new information to be stored into the memory is called a *write* operation.
- The particular memory word among the many available is selected by the memory address during the transfer.
- Consider a memory unit that receives the address from a register, called the address register, symbolized by AR.
- The data is transferred to another register, called the data register, symbolized by DR.
- The memory read operation can be stated as follows

  Read: DR ←M [AR]
- Here data is transferred into DR from the memory location specified by AR.
- The memory write operation can be stated as follows

  Write: M [AR]← DR
- Here write operation transfers the content of a data register to a memory word M specified by the address in AR.

The micro operations most often encountered in digital computers are classified into four categories:

1. Register transfer micro operations transfer binary information from one register to another.
2. Arithmetic micro operations perform arithmetic operation on numeric data stored in registers.
3. Logic micro operations perform bit manipulation operations on non-numeric data stored in registers
4. Shift micro operations perform shift operations on data stored in registers

1.4 ARITHMETIC MICRO OPERATIONS

- The basic arithmetic microoperations are addition, subtraction, increment and decrement.

Add Microoperation:

$$R3 \leftarrow R1 + R2$$

- The above statement states that the contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.

Subtract Microoperation:

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

- In the above statement $\overline{R2}$ is the symbol for the 1's complement of R2. Adding 1 to the 1's complement produces the 2's complement. Adding the contents of R1 to the 2's complement of R2 is equivalent to R1 – R2.
- The increment and decrement microoperations are symbolized by plus-one and minus-one operations, respectively.

Table 5: Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow \overline{R2}$	Complement the contents of R2 (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's complement of R2 (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one
$R1 \leftarrow R1 - 1$	Decrement the contents of R1 by one

1.4.1 Binary Adder

- To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the arithmetic addition.
- The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

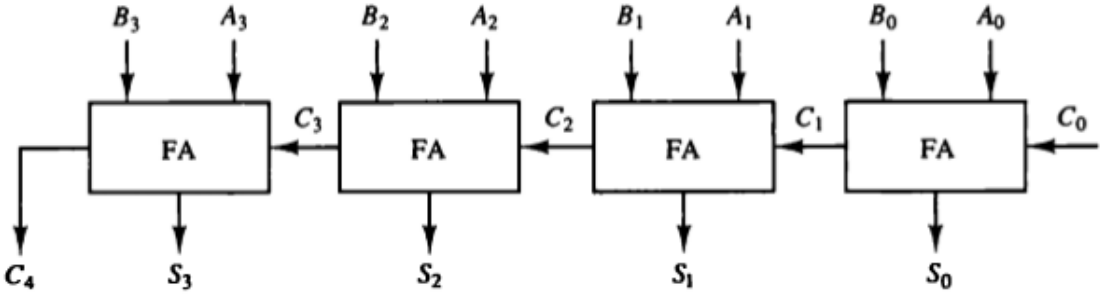


Figure 9: 4-bit binary adder

- The S outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders. The output carry from each full-adder is connected to the input carry of the next-higher-order full-adder.

1.4.2 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.
- A 4-bit adder-subtractor circuit is shown in the following figure.

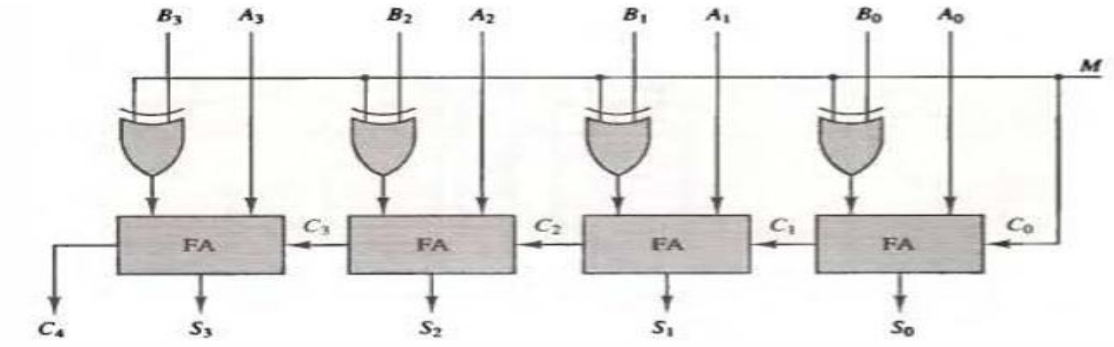


Figure 10: Adder-subtractor

- In the above figure, 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 B. When M = 0, we have B⊕0 = B. The full-adders receive the value of B, the input carry is 0, and the circuit performs A+B.
- When M = 1, we have B⊕1 = B' and C_0 = 1. The B inputs are all complemented and 1 is added through the input carry. The circuit performs the operation A+2's complement of B.

1.4.3 Binary Incrementer

- The increment microoperation adds one to a number in the register.
- The diagram of a 4-bit incrementer circuit is shown below. One of the inputs of the least significant half-adder (HA) circuit is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.

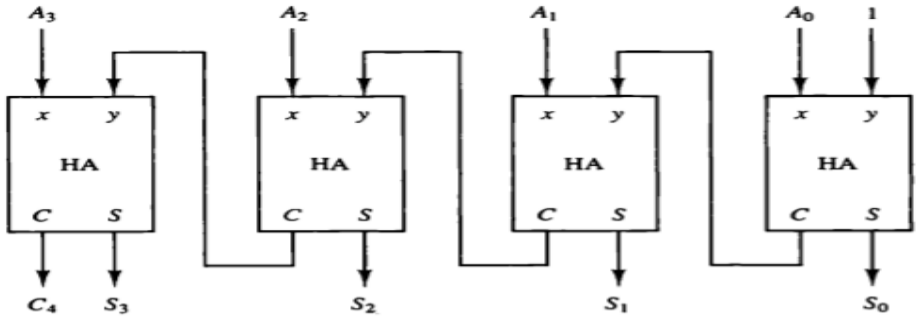


Figure 11: 4-Bit Binary Incrementer

- The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder. The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output S_0 through S_3.

1.4.4 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 as shown in the following figure. 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 full adder. Each of the four inputs from B are connected to the data inputs of the multiplexers.
- The four multiplexers are controlled by two selection inputs, S_1 and S_0.

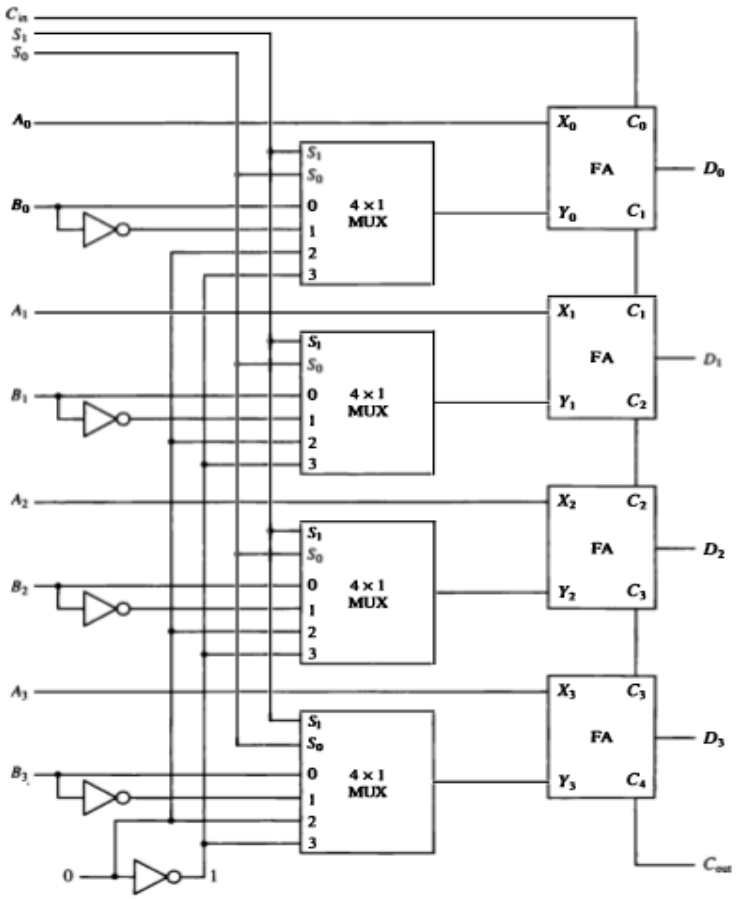


Figure 12: 4-bit arithmetic circuit

- 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.
- The output of the binary adder is calculated from the following arithmetic sum:

$$D = A + Y + C_{in}$$

- where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. C_{in} is the input carry, which can be equal to 0 or 1.
- 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 the following table.

Table 6: 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	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	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

1.5 LOGIC MICROOPERATIONS

- 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 on the contents of two registers R 1 and R2 is symbolized by the statement.

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

- The above specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1.
- As a numerical example, assume that each register has four bits. Let the content of R1 be 1010 and the content of R2 be 1100. The exclusive-OR microoperation stated above symbolizes the following logic computation:

1010 Content of R1

1100	Content of R2
0110	Content of R1 if P = 1

- The symbol **V** will be used to denote an **OR** microoperation and the symbol **Λ** to denote an **AND** microoperation. The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.

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.

Table 7: 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

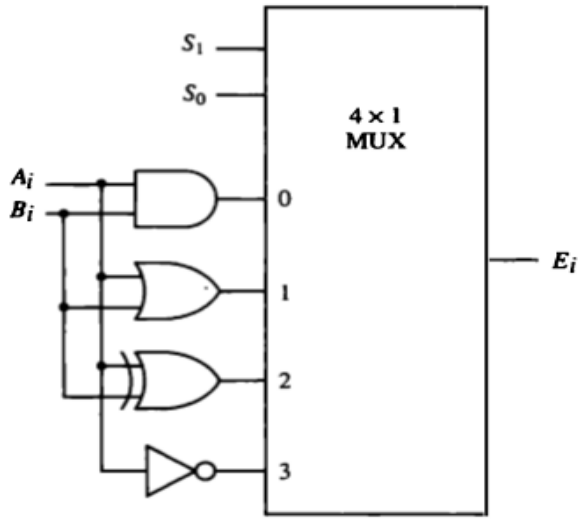
- In the following table, each of the 16 columns F_0 through F_{15} represents a truth table of one possible Boolean function for the two variables x and y. Note that the functions are determined from the 16 binary combinations that can be assigned to F.

Table 8: 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

1.5.1 Hardware Implementation for Logic Microoperations

- 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.
- The following figure shows one stage of a circuit that generates the four basic logic mnicrooperations. It consists of four gates and a multiplexer.
- 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.



(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

Figure 13: One stage of logic circuit

1.5.2 Applications of logic microoperations

1.5.2.1. selective-set

- The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B. The following numerical example clarifies this operation:

1010	A before
1100	B (logic operand)
1110	A after

1.5.2.2. selective-complement

- The selective-complement operation complements bits in A where there are corresponding l's in B. It does not affect bit positions that have 0's in B.

  For example:

1010	A before
1100	B (logic operand)
0110	A after

1.5.2.3. Selective-clear

- The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B.

  For example:

1010	A before
1100	B (logic operand)
0010	A after

1.5.2.4. mask

- The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0'sin B. The mask operation is an AND micro operation as seen from the following numerical example:

1010	A before
1100	B (logic operand)
1000	A after masking

1.5.2.5. Insert

- The insert operation inserts a new value into a group of bits.
- This is done by first masking the bits and then ORing them with the required value.
- For example, an A register contains eight bits, 0110 1010. To replace the four leftmost bits by the value 1001.

First mask the four unwanted bits.

0110 1010	A before masking
0000 1111	B (mask)
0000 1010	A after masking

Now insert the new value.

0000 1010	A before insert
1001 0000	B (insert)
1001 1010	A after insertion

**The mask operation is an AND microoperation and the insert operation is an OR microoperation.

1.5.2.6. clear

- The clear operation compares the words in A and B and produces an all 0's result, if the two numbers are equal.
- This operation is achieved by an exclusive-OR microoperation as shown by the following example.

1010	A
1010	B
0000	A ←A⊕B

1.6 SHIFT MICROOPERATIONS

- Shift rnicrooperations are used for serial transfer of data.
- They are also used in conjunction with arithmetic, logic, and other data-processing operations.
- The contents of a register can be shifted to the left or the right.
- There are three types of shifts:
    1. Logical Shift
    2. Circular Shift
    3. Arithmetic Shift

1.6.1. Logical Shift

- A logical shift is one that transfers 0 through the serial input to fill the vacancy created by shift operation.
- The symbols *shl* is used for logical shift-left rnicrooperation.

  Shr is used for shift-right rnicrooperation.

  shl: Example: R1 ←shl R1

  the above statement left shifts the content of register R1 by 1-bit.

Example: R1 = 0110

After performing shift left, R1 has the content 1100

shr: Example: R1 ←shr R1

the above statement right shifts the content of register R1 by 1-bit.

Example: R1 = 1100

After performing shift left, R1 has the content 0110

1.6.2. Circular Shift

- The circular shift also known as a rotate operation.
- It circulates the bits of the register around the two ends without loss of information.
- The symbols *cil* used for logical circular shift-left rnicrooperation.

cir used for circular shift-right rnicrooperation.

cil: Example: R1 ←cil R1

the above statement specifies circular shift left of the content of register R1.

Here all the bits are shifted one bit position to LEFT, the Left most bit (MSB) was circulated to Right most bit (LSB).

Example: R1 = 1011

After performing circular shift left, R1 has the content 0111

cir: Example: R1 ←cir R1

the above statement specifies circular shift right of the content of register R1.

Here all the bits are shifted one bit position to RIGHT, the Right most bit (LSB) was circulated to Left most bit (MSB).

Example: R1 = 1011

After performing circular shift right, R1 has the content 1101

1.6.3. Arithmetic Shift

- An arithmetic shift is a microoperation that shifts a signed binary number to the left or right.
- After the shift microoperation the sign of the number is to be restored.
- The symbols ash*l* used for logical arithmetic shift-left rnicrooperation.

ashr used forarithmetic shift-right rnicrooperation

ashl: Example: R1 ←ashl R1

the above statement specifies arithmetic shift left of the content of register R1

Here all the bits except the MSB, shift one bit position to LEFT. The second Left most bit was discarded and Right most bit (LSB) was loaded by 0.

Example: R1 = 1011

After performing arithmetic shift left, R1 has the content 1110

ashr: Example: R1 ←ashr R1

the above statement specifies arithmetic shift right of the content of register R1

Here all the bits shifted one bit position to RIGHT. The Left most bit (MSB) remains same.

Example: R1 = 1011

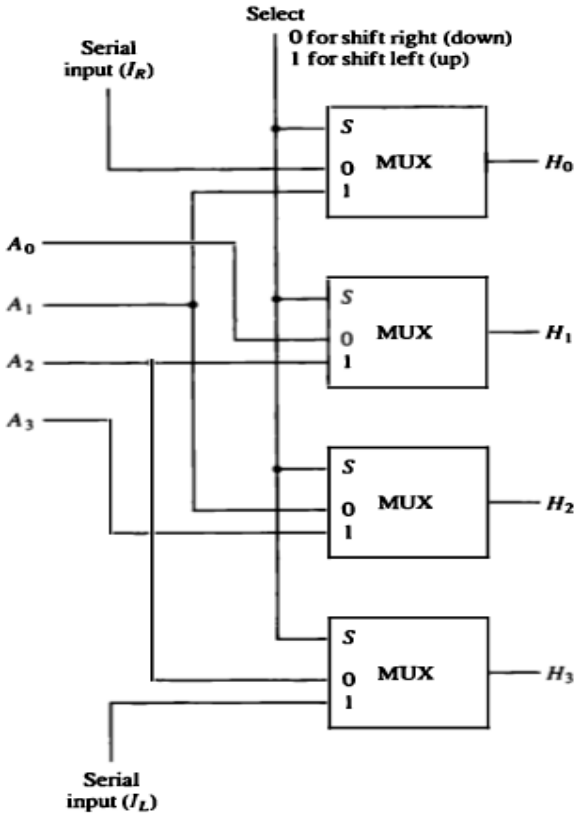
After performing arithmetic shift right, R1 has the content 1101

Table 9: Shift Microoperations

Symbolic designation	Description
R ← shl R	Shift-left register R
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left register R
R ← cir R	Circular shift-right register R
R ← ashl R	Arithmetic shift-left R
R ← ashr R	Arithmetic shift-right R

1.6.4 Hardware Implementation for Shift Microoperations

- A combinational circuit shifter can be constructed with multiplexers as shown in figure.



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 14: 4-bit combinational circuit shifter

** Here H_0 bit is considered as MSB and H_3 bit is considered as LSB.

- The 4-bit combinational circuit shifter uses four multiplexers, each of 2X1.
- 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 above function table shows which input goes to each output after the shift.
- A shifter with n data inputs and n data outputs requires n multiplexers each of 2 X 1.

1.6.5 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.
- To perform a microoperation, the contents of specified registers are placed in the inputs of the ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register.
- 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 arithmetic, logic, and shift circuits can be combined into one ALU with common selection variables.
- One stage of an arithmetic logic shift unit is shown in the following figure.

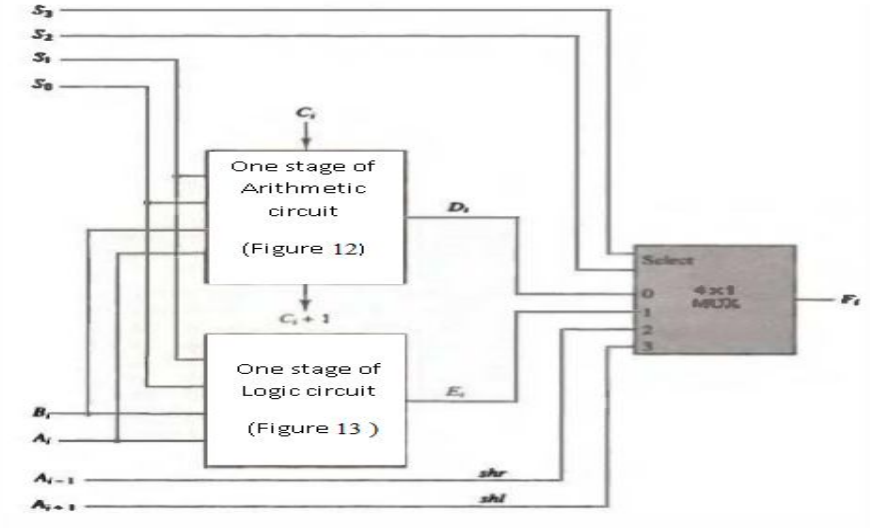


Figure 15: One stage of Arithmetic Logic Shift Unit

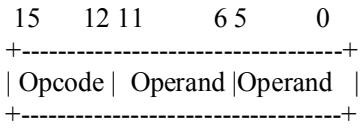
- Inputs A_i and B_i are applied to both the arithmetic and logic units.
- A particular microoperation is selected with inputs S_1 and S_0.
- A 4 x 1 multiplexer at the output chooses between an arithmetic output in D_i and a logic output in E_i.
- The data in the multiplexer are selected with inputs 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 the above figure provides eight arithmetic microoperation, four logic microoperations, and two shift microoperations.
- 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 arithmetic operations only.
- The first eight are arithmetic microoperations, which are selected with $S_3S_2 = 00$.
- The next four are logic microoperations, which are selected with $S_3S_2 = 01$.
- The input carry has no effect during the logic microoperations and is marked with don't-care x.
- The last two operations are shift microoperations and are selected with $S_3S_2 = 10$ for shift right microoperation and S_3S_2=and 11 for shift left microoperation. The other three inputs have no effect on the shift.

Table 10: Function Table for Arithmetic Logic Shift Unit

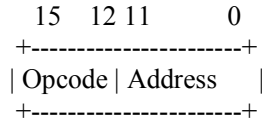
Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction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

1.7 INSTRUCTION CODES

- An instruction code is a group of bits which instructs the computer to perform certain operation.
- Instructions are encoded as binary *instruction codes*. Each instruction code contains a *operation code*, or *opcode*, which designates the overall purpose of the instruction (e.g. add, subtract, move, input, etc.).
- The number of bits allocated for the opcode determines how many different instructions the architecture supports.
- In addition to the opcode, many instructions also contain one or more operands, which indicate where in registers or memory the data required for the operation is located.
- For example, an *add* instruction requires two operands, and a *not* instruction requires one.



- Suppose all instruction codes of a hypothetical accumulator-based CPU are exactly 16 bits. A simple instruction code format could consist of a 4-bit operation code (opcode) and a 12-bit memory address.



1.7.1 Stored Program Organization

- The simplest way to organize a computer is to have one process register and an instruction code formats with two parts.
- The first part specifies the operation to be performed and the second specifies the address.
- The instructions are stored in one section of the memory and the data is stored in the another section.
- The figure considers the memory of size 4096 x 16. The number of Address lines required to represent the 4096 words are 12 because $4096=2^{12}$. The number of data lines required are 16.

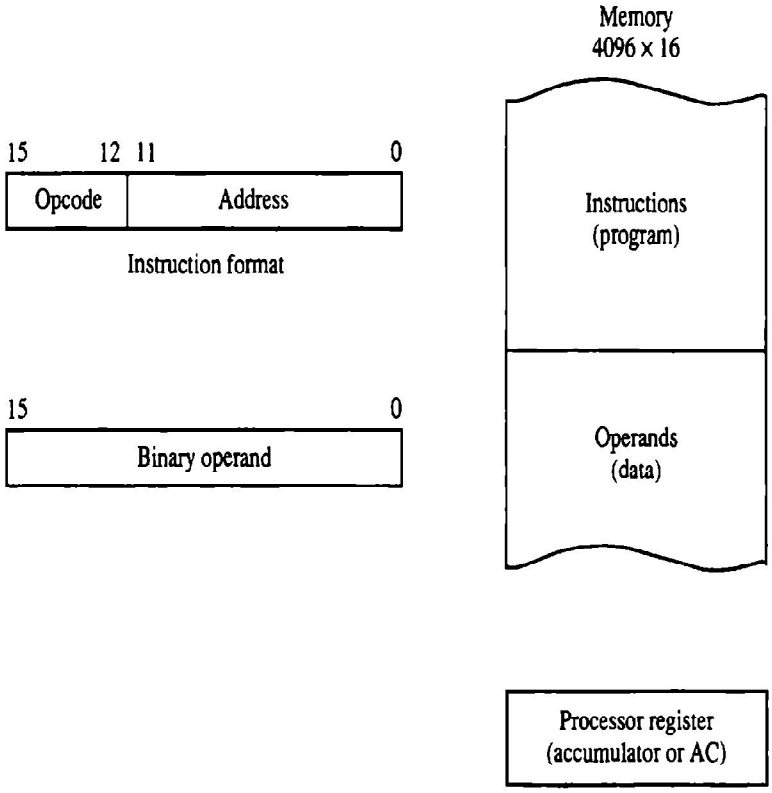


Figure 16: Stored program organization

1.8 INSTRUCTION FORMAT

- A 16 bit instruction code format could consist of a 4-bit operation code (opcode) and a 12-bit memory address, where the 4 bits are divided into 2 parts as shown in the figure below.
- The bits 12-14 represent the operation code and the 15th bit which is represented with I states whether the given address is Direct or Indirect.

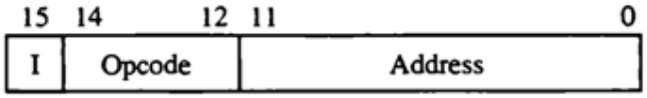


Figure 17: Instruction format

- If the value of I = 0 then the given address is Direct address and if the value of I = 1 then the given address is Indirect address.
- Direct : Instruction code contains address of operand.
- Immediate : Instruction code contains operand
- Indirect : Instruction code contains address of address of operand.
- Effective address = actual address of the data in memory.

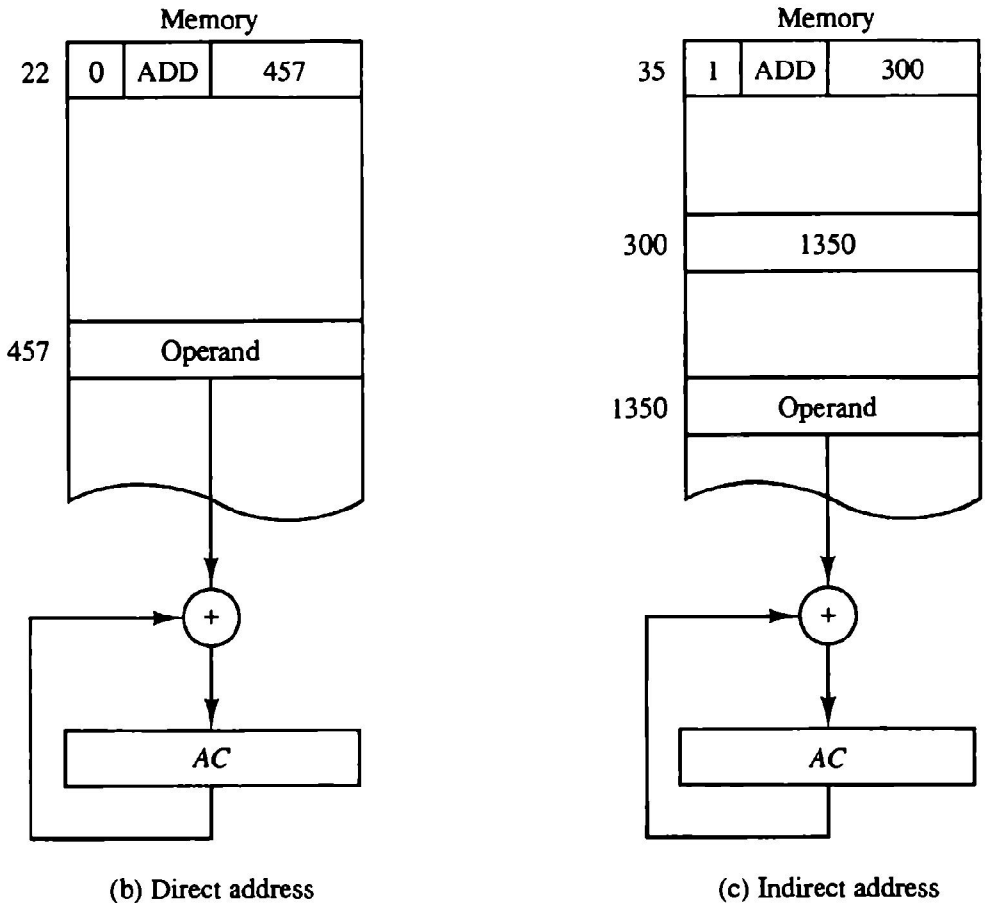


Figure 18: Demonstration of direct and indirect address

Example: The following examples use the following piece of computer memory.

1.9 INSTRUCTION CYCLE

The program is executed in the computer by going through a cycle for each instruction. 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 following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

After the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered or no further instructions to be executed.

1.9.1 Fetch and Decode

- Initially, the program counter PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T_0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on.
- The Microoperations for the fetch and decode phases can be specified by the following register transfer statements.

  T_0: AR ← PC

  T_1: IR ←M[AR], PC ← PC + 1

  T2: D0, • • • , D7 ← Decode IR(12-14), AR ← IR(0-11), 1 ← IR(l5)
- Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T_0.
- The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T_1. At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.
- At time T_2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR .
- The first two register transfer statements are implemented in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection:
  1. Place the content of PC onto the bus by making the bus selection inputs S_2 S_1 S_0 equal to 010.
  2. Transfer the content of the bus to AR by enabling the LD input of AR. To implement this it is necessary to use timing signal T_1 to provide the following connections in the bus system.

     T_1: IR ← M[AR], PC ← PC + 1

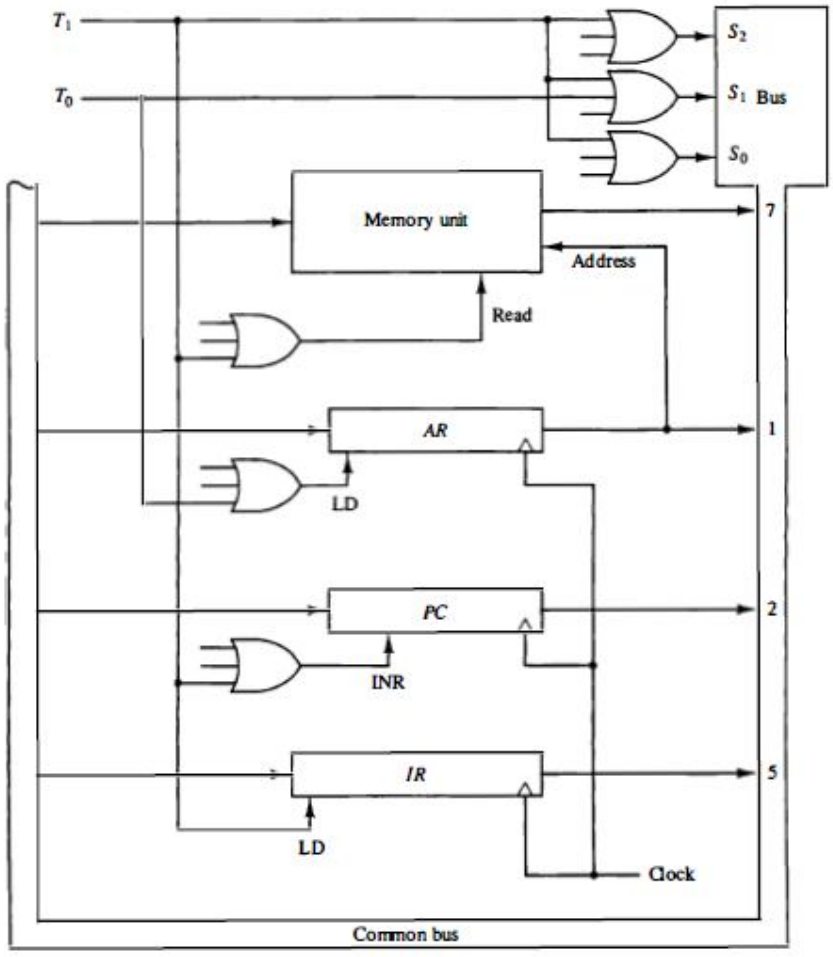


Figure 19: Register Transfers for the Fetch Phase

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to IR by enabling the LD input of IR .
4. Increment PC by enabling the INR input of PC.

1.9.2 Determine the Type of Instruction

- Decoder output D7= 1 and I = 1 then it is called I/O reference instruction.
- If D7 = 1, and I = 0 the instruction must be a register-reference.
- If D7 = 0, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.
- If D7 = 0 and I = 1, we have a memory reference instruction with an indirect address.
- If D7 = 0 and I = 0, we have a memory reference instruction with a direct address.

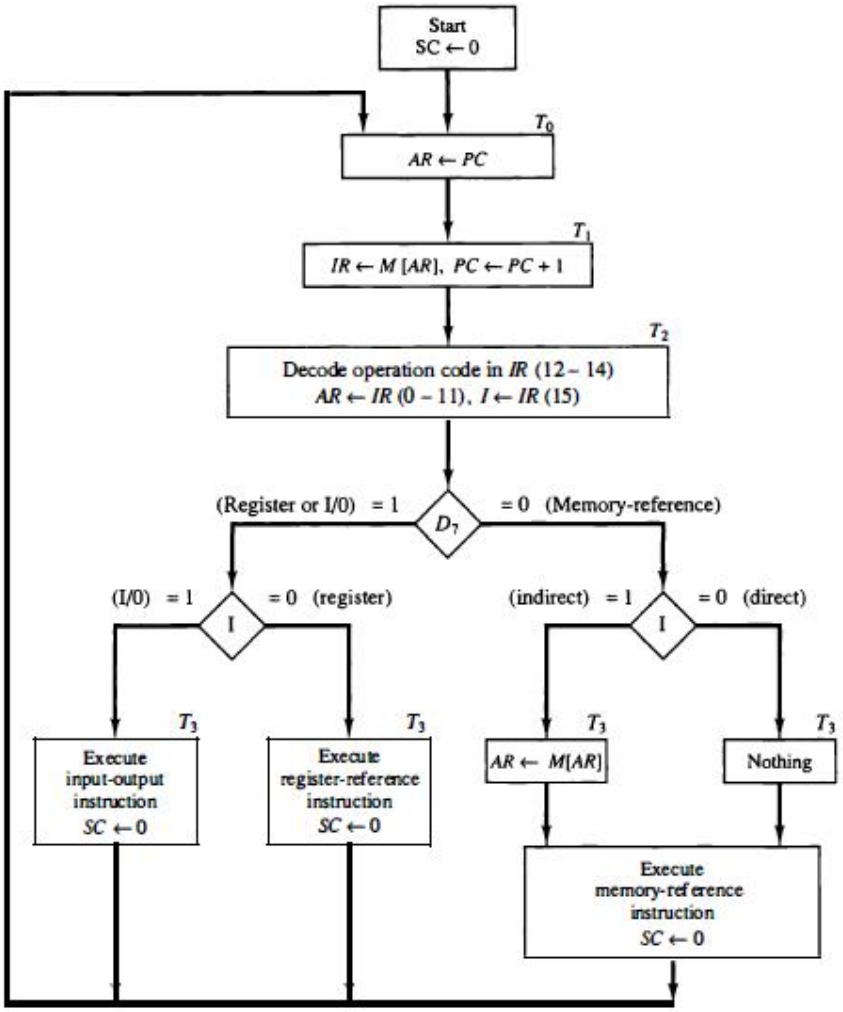


Figure 20: Flow Chart for Instruction Cycle (Initial Configuration)

The micro operation for the indirect address condition can be symbolized by the register transfer statement.

$$AR \leftarrow M[AR]$$

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows:

$D_7' IT_3$: $AR \leftarrow M[AR]$

$D_7' I' T_3$: Nothing

$D_7 I' T_3$: Execute a register-reference instruction

$D_7 I T_3$: Execute an input-output instruction

1.10 COMPUTER INSTRUCTIONS

The basic computer has three instruction code formats:

1. Memory Reference Instructions
2. Register Reference Instructions
3. Input / Output Instructions

1.10.1. Memory Reference Instructions

In Memory reference instruction:

- First 12 bits(0-11) specify an address.
- Next 3 bits specify operation code (opcode) and can range from 000 to 110.
- Left most bit specify the addressing mode I
- I = 0 for direct address
- I = 1 for indirect address
- The address field is denoted by three x's (in hexadecimal notation) and is equivalent to 12-bit address.
- When I = 0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is zero (0).
- When I = 1 the last four bits of an instruction have a hexadecimal digit equivalent from 8 to E since the last bit is one (1).

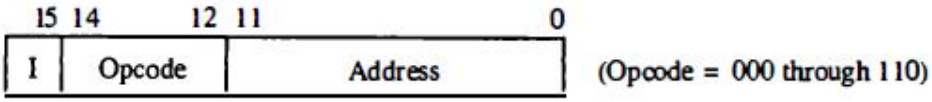


Figure 21: Memory - reference instruction format

Table 11: Basic Computer Instructions for Memory Reference Instructions

Symbol	Hexadecimal code		Description
	I = 0	I = 1	
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

1.10.2. Register Reference Instructions

In Register Reference Instruction:

- First 12 bits (0-11) specify the register operation.

- The next three bits equals to 111 specify opcode.
- The last mode bit of the instruction is 0.
- Therefore, left most 4 bits are always 0111 which is equal to hexadecimal 7.

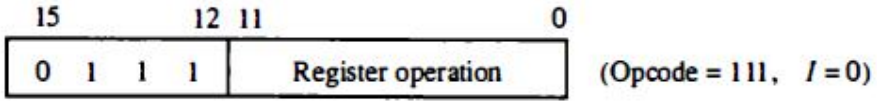


Figure 22: Register - reference instruction format

Table 12: Basic Computer Instructions for Register - reference instructions

Symbol	Hexadecimal code	Description
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 is AC is negative
SZA	7004	Skip next instruction is AC is 0
SZE	7002	Skip next instruction is E is 0
HLT	7001	Halt computer

1.10.3. Input / Output Instructions

In I/O Reference Instruction:

- First 12 bits (0-11) specify the I/O operation.
- The next three bits equals to 111 specify opcode.
- The last mode bit of the instruction is 1.
- Therefore, left most 4 bits are always 1111 which is equal to hexadecimal F.

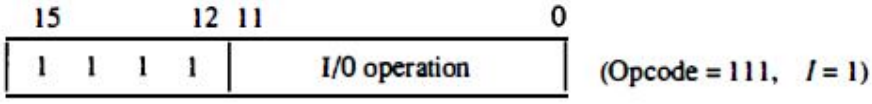


Figure 23: Input - output instruction format

Table 13: Basic Computer Instructions for Input - output instructions

Symbol	Hexadecimal code	Description
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

1.11 MEMORY-REFERENCE INSTRUCTIONS

- In order to specify the micro operations needed for the execution of each instruction, it is necessary that the function that they are intended to perform be defined precisely.
- The function of the memory-reference instructions can be defined precisely by means of register transfer notation. Table given below lists the seven memory-reference instructions. 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 the table.
- The data must be read from memory to a register where they can be operated on. We will see the operation of each instruction and list the control functions and micro operations needed for their execution.

Table 14: Memory Reference Instructions

Symbol operation	Decoder	Symbolic description
AND	D_o	AC ← AC ∧ M[AR]
ADD	D_1	AC ← AC + M[A R] , E ← C_{OUT}
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC , PC ←AR + 1
ISZ	D_6	M[AR] ← M[AR] + 1 ,if M[AR] + 1 = 0 then PC ← PC + 1

- After the instruction is fetched from memory and decoded, only one output of the operation decoder will be active, and that output determines the sequence of micro operations that the control follows during the execution of a memory-reference instruction.

AND to AC

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

D_0T_4: DR ← M [AR]

D_0T_5: AC ← AC ∧ DR, SC ← 0

- Two timing signals are needed to execute the instruction.

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 bit of accumulator). The micro operations needed to execute this instruction are

D_1T_4: DR ← M [AR]

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0

LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC. The micro operations needed to execute this instruction are

D_2T_4: DR ← M [AR]

D_2T_5: AC ← DR, SC ← 0

- As we know that there is no direct path from the bus into AC. The adder and logic circuit receive information from DR which can be transferred into AC. Therefore, it is necessary to read the memory word into DR first and then transfer the content of DR into AC.

STA: Store AC

- This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can complete the execution of this instruction in 1 timing signal.

D_3T_4: M [AR] ←AC, SC ←0

BUN: Branch Unconditionally

- It allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally to the location specified by effective address. The instruction is executed with one micro operation:

D_4T_4: PC ←AR, SC ←0

Branch and Save Return Address (BSA)

- This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.
- The effective address plus one is then transferred to PC to serve as the address of the first instruction in the subroutine.

M [AR] ←PC, PC ←AR + 1

- The BSA instruction performs the following numerical operation:

M[135] ←21, PC ←135 + 1 = 136

- The result of this operation is shown in part (b) of the figure. The return address 21 is stored in memory location 135 and control continues with the subroutine program starting from

address 136. The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.

- The BSA instruction performs the function usually referred to as a subroutine call. The indirect BUN instruction at the end of the subroutine performs the function referred to as a subroutine return.

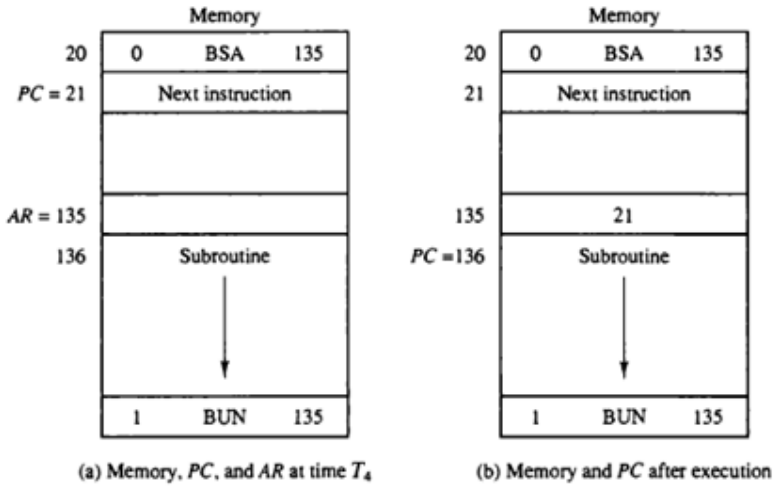


Figure 24: Example of BSA Instruction Execution

- It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer.
- To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two micro operations:

D_5T_4: M [AR] ←PC, AR ←AR + 1

D_5T_5: PC ←AR, SC ←0

- Timing signal T_4 initiates a memory write operation, places the content of PC onto the bus, and enables the INR input of AR.
- The memory write operation is completed and AR is incremented by the time the next clock transition occurs. The bus is used at T5 to transfer the content of AR to PC.

Increment and Skip if Zero (ISZ)

- This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.
- The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by 1 in order to skip the next instruction in the program.

- 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 into memory. This is done with the following sequence of micro operations:

D_6T_4: DR ←M [AR]

D_6T_5: DR ←DR + 1

D_6T_6: M [AR] ←DR, if (DR = 0) then (PC ←PC + 1), SC ←0

1.11.1 Control Flowchart

- A flowchart showing all micro operations for the execution of the seven memory- reference instructions is shown in Fig. given below.
- The control functions are indicated on top of each box. The micro operations that are performed during time T_4, T_5, or T_6, depending on the opcode value. This is indicated in the flowchart by six different paths, one of which the control takes after the instruction is decoded.
- The sequence counter SC is cleared to 0 during the last timing signal in each case. This causes a transfer of control to timing signal T_0 to start the next instruction cycle.
- Note that we need only seven timing signals to execute the longest instruction (ISZ) requiring the 3-bit sequence counter.

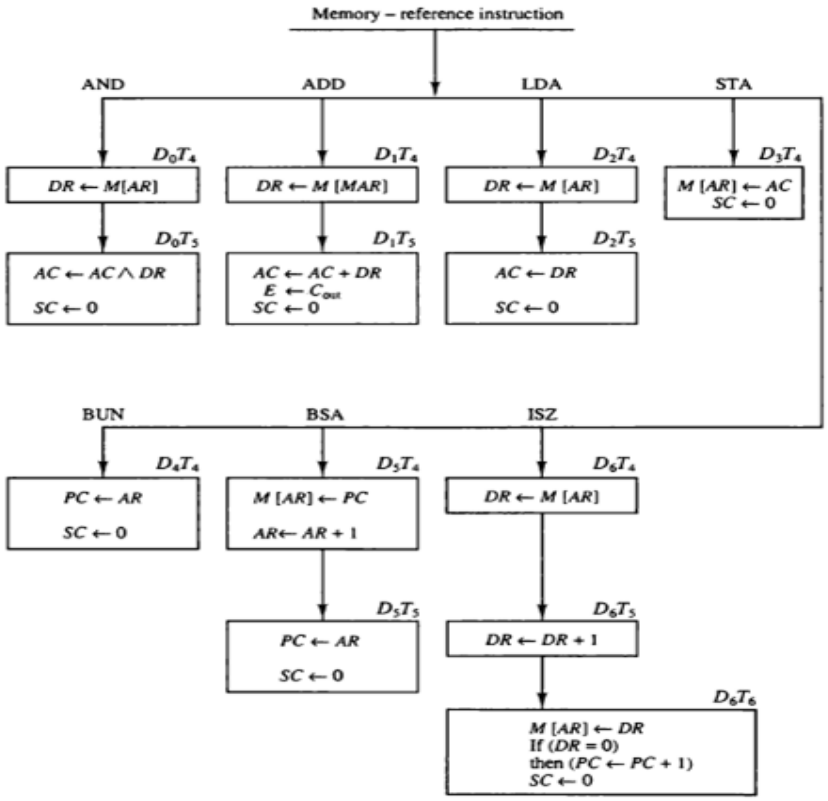


Figure 25: Flowchart for Memory Reference Instructions

Unit - I

Assignment-Cum-Tutorial Questions

Section - A

1. Which language is termed as the symbolic depiction used for indicating the Operations?

A. Random transfer language B. Register transfer language [ ]

C. Arithmetic transfer language D. All of these

2. Micro operation is shown as? [ ]

A. R1← R2 B. R1 → R2 C. Both A and B D. None of these

3. Write the RTL code for transferring the contents of register R1 into R2, when p=1.

4. The register that includes the address of the memory unit is termed as the _____ [ ]

A. MAR B. PC C. IR D. None of these

5. Operation to transfer contents into memory is termed as _____ [ ]

A. Read B. Write C. Both A & B D. None of these

6. LOAD R2, 30FF is_____ type of instruction? [ ]

A. Arithmetic and Logical instruction B. Control instruction

C. Data transfer instruction D. None of the above

7. In 3 state buffer, two states act as signals equal to? [ ]

A. Logic 0 B. Logic 1 C. Both A & B D. None of these

8. In 3 state buffer third position termed as high impedance state which acts as? [ ]

A. Open circuit B. Close circuit C. Both A & B D. None of these

9. Which operations are used for addition, subtraction, increment, decrement and complement function? [ ]

A. Bus B. Memory transfer

C. Arithmetic operation D. All of these

10. What are the operations that a computer performs on the data stored in a register? [ ]

A. Register transfer B. Arithmetic C. Logical D. All of these

11. Which operation places memory address in memory address register and data in MDR? [ ]

A. Memory read B. Memory write C. Both A & B D. None of these

12. Which operation is extremely useful in serial transfer of data? [ ]

A. Logical micro operation B. Arithmetic micro operation

C. Shift micro operation D. None of these

13. A group of bits that tell the computer to perform a specific operation is known as___ [ ]

A. Operation code B. Micro-operation

C. Accumulator D. Register

14. How many bits of opcode is required to implement a CPU with 5 arithmetic and logical instructions, 2 control instructions, and 4 data transfer instructions? [ ]

A. 1 B. 2 C. 3 D. 4

15. What is the combination of I and Opcode bits for I/O instructions?

16. The CPU of a Computer takes instruction from the memory and executes them. This process is called as_____? [ ]

A. Load cycle B. Time sequence

C. Fetch-execute cycle D. None of these

17. A CPU has 24-bit instructions. A program starts at address 300 (in decimal). Which one of the following is a legal program counter (all values in decimal)? **(GATE 2006)** [ ]

A. 400 B. 500 C. 600 D. 700

18. Assume a CPU takes 17 cycles in worst case to execute an instruction. Number of cycles required to execute the current instruction is 12.If an interrupt occurs during the execution of current instruction, then after how many cycles it will be recognized? [ ]

A. 17 B. 11 C. 12 D. 17+2

Section - B

1. Explain how various registers and memory are connected using a common bus with diagram.
2. Design a bus system for connecting 4 registers each of size 8 bits.
3. Explain various Arithmetic Micro operations with example.
4. Design a 6-bit Adder/Subtractor circuit.
5. Design a 4-bit Incrementer Circuit.
6. Construct 6-bit arithmetic circuit.
7. Describe various Logical Micro operations.
8. What are different shift microoperations? Explain with an example.
9. Explain the working of the circuit that performs Logic operations.
10. Explain different types of instructions.
11. Explain the life cycle of an instruction with a suitable flow chart.

Unit - II

CPU and Micro Programmed Control

Objective:

- To familiarize the concept of Addressing Modes, Control memory, Hard wired control, Micro programmed control.

Syllabus:

Central Processing unit: Introduction, instruction formats, addressing modes. Control Memory, address sequencing, design of control unit - hard wired control, micro programmed control.

Learning Outcomes:

At the end of the unit student will be able to:

1. Differentiate micro-programmed and hard-wired control units

Learning Material

INSTRUCTION FORMATS

- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into 3 fields with each field consisting of a group of bits.
    a. **Operation code** field that specifies the operation to be performed
    b. **Address** field that designates a memory address or processor register where the operand is present
    c. **Mode** field that specifies the way the operand or the effective address is to be determined.
- Other special fields are sometimes employed under certain circumstances.
- Instruction formats are concerned with the address fields in an instruction.
- Operations specified by computer instructions are executed on some data stored in memory or registers.
- Operands residing in memory are specified by their memory address.
- Operands residing in registers are specified with a register address.
- A CPU with 16 processor registers labeled R_0 through R_{15} will have a register address field of 4 bits.

- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.

i. Single accumulator organization

- All operations are performed in combination with accumulator. It requires only one address field.
- ADD X, where X is the address of the operand. The ADD instruction results in the operation
- AC←AC+M[X]

ii. General register organization

- It requires two or three register address fields

  ADD R1,R2,R3 resulting in the operation R1←R2+R3.
- The number of address fields in the instruction can be reduced from three to two if the destination register is one of the source registers.

  ADD R1,R2 resulting in operation R1←R1+R2. These instructions need two address fields to specify the source and the destination.

iii. Stack organization

- It has PUSH and POP instructions which require no address field.
- Consider the instruction ADD, this operation has the effect of popping the top two elements from the stack, adding the numbers, and pushing the sum into the stack.

Let us see the assembly language programs in different instruction formats that evaluates X=(A+B)*(C+D) as follows.

Figure 1: Four common Instruction formats

THREE ADDRESS INSTRUCTIONS

- Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

```
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
```

Merit: it results in short programs when evaluating arithmetic expressions.

Demerit: The binary-coded instructions require too many bits to specify three addresses.

TWO ADDRESS INSTRUCTIONS

- Each address field can specify either a processor register or a memory word.

```
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
```

The MOV instruction moves or transfers the operands to and from memory and processor registers.

ONE ADDRESS INSTRUCTIONS

- One address instructions use an implied accumulator (AC) register for all data manipulations.

```
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
```

- All operations are done between accumulator register and a memory operand. T is the address of the temporary memory location required for storing the intermediate result.

ZERO ADDRESS INSTRUCTIONS

- A stack organized computer does not use an address field in the instructions. The PUSH and POP instructions, however need an address field to specify the operand that communicates with the stack.

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

- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse polish (Postfix) notation.

ADDRESSING MODES

- In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and the mode of the instruction.
- Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.
- An example of an instruction format with a distinct addressing mode field is shown in the below figure 2.

Figure 2: Instruction format with mode field

- The operation code specifies the operation to be performed.
- The mode field is used to locate the operands needed for the operation.
- There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register.
- The instruction may have more than one address field, and each address field may be associated with its own particular addressing mode.

- There are two modes that need no address field at all. These are the implied and immediate modes.

1. Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction.

- **For example**, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- In fact, all register reference instructions that use an accumulator are implied-mode instructions.

Example: Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

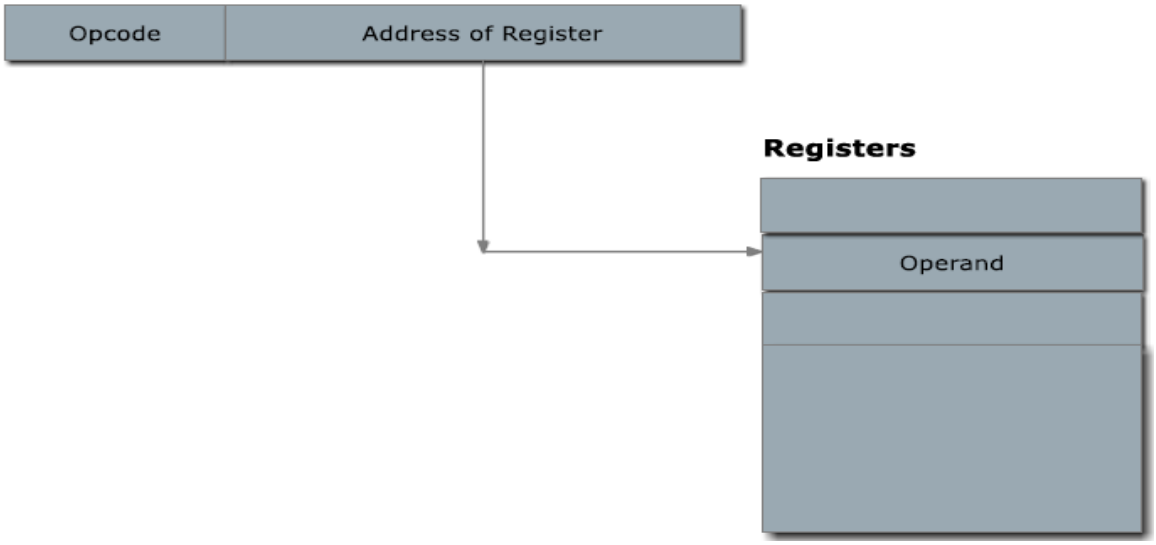
CMA, CME, CLE, CLA

2. Immediate Mode: In this mode the operand is specified in the address part of the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field. Immediate-mode instructions are useful for initializing registers to a constant value.

LD #20

3. Register Mode: In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.

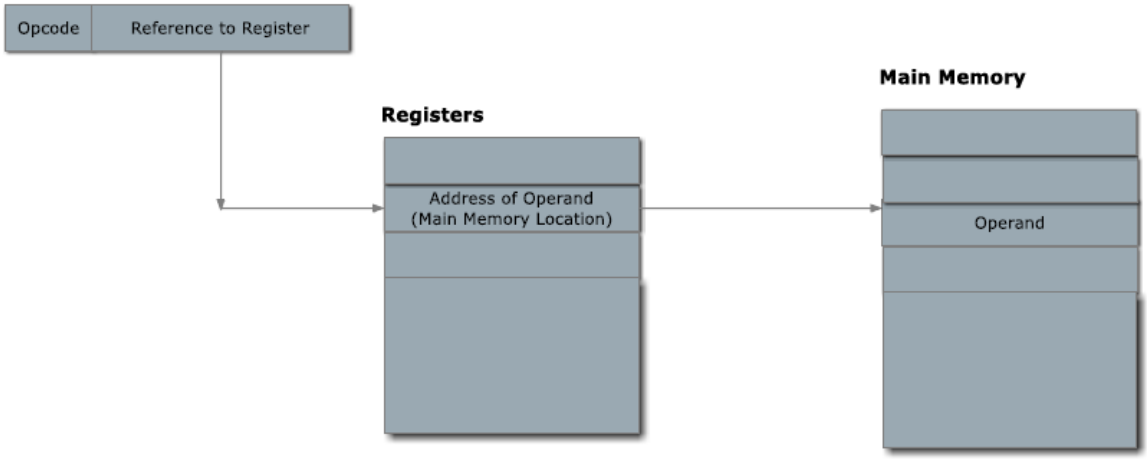
LD R



4. Register Indirect Mode:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address for the operand is placed in the processor register with a previous instruction.
- A reference to the register is then equivalent to specifying a memory address.
- The advantage of a register indirect mode instruction is that the address field of the instruction sues fewer bits to select a register than would have been required to specify a memory address directly.

LD (R) OR **LD** @R



5. Auto increment or Auto decrement Mode:

- This is similar to the register indirect mode except that the register value is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.

LD (R) + Auto increment

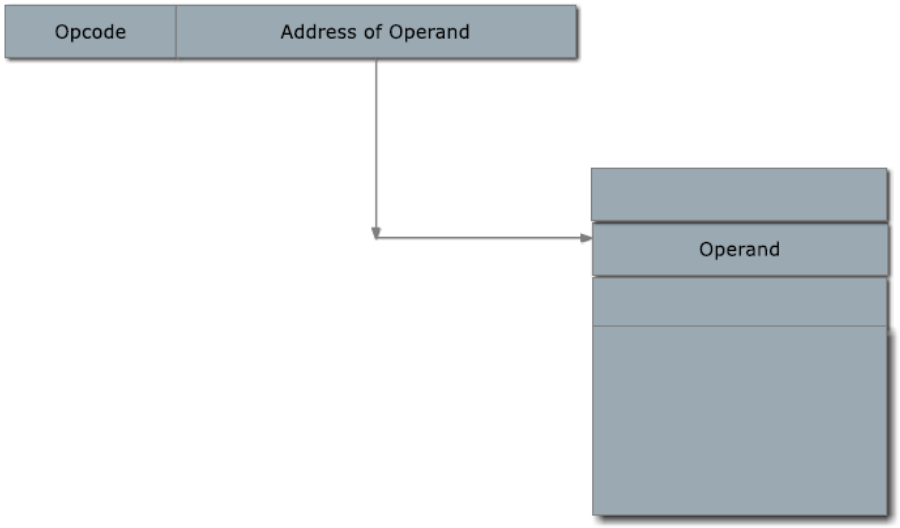
LD (R) - Auto decrement

6. Direct Address Mode:

- In this mode the effective address is equal to the address part of the instruction. The operand resides in memory.

- In a branch-type instruction the address field specifies the actual branch address.

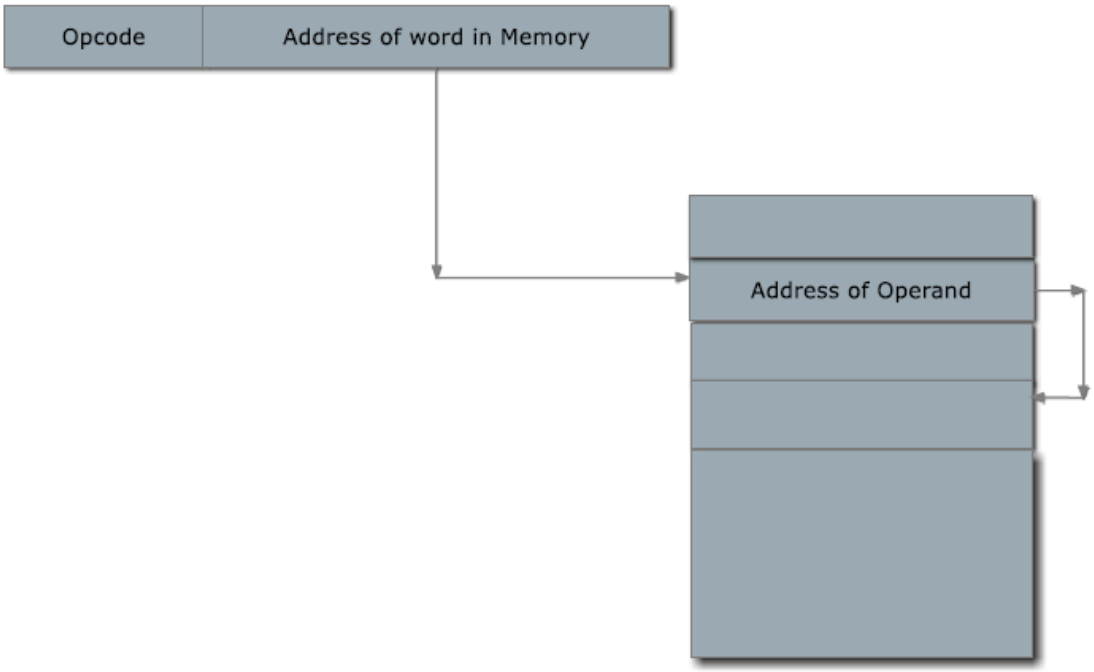
LD 10



7. Indirect Address Mode:

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

LD (10)



8. Relative Address Mode:

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.

LD $5

9. Indexed Addressing Mode:

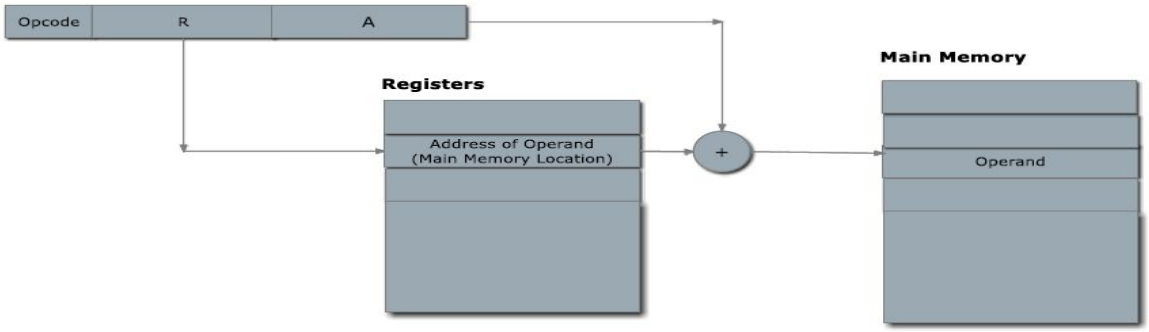
- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory.

LD 5(IR)

10. Base Register Addressing Mode:

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.
- The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

LD 5(BR)



Displacement addressing mode (Relative Addressing, Base Register Addressing, Indexed Addressing)

Numerical Example of Addressing Modes:

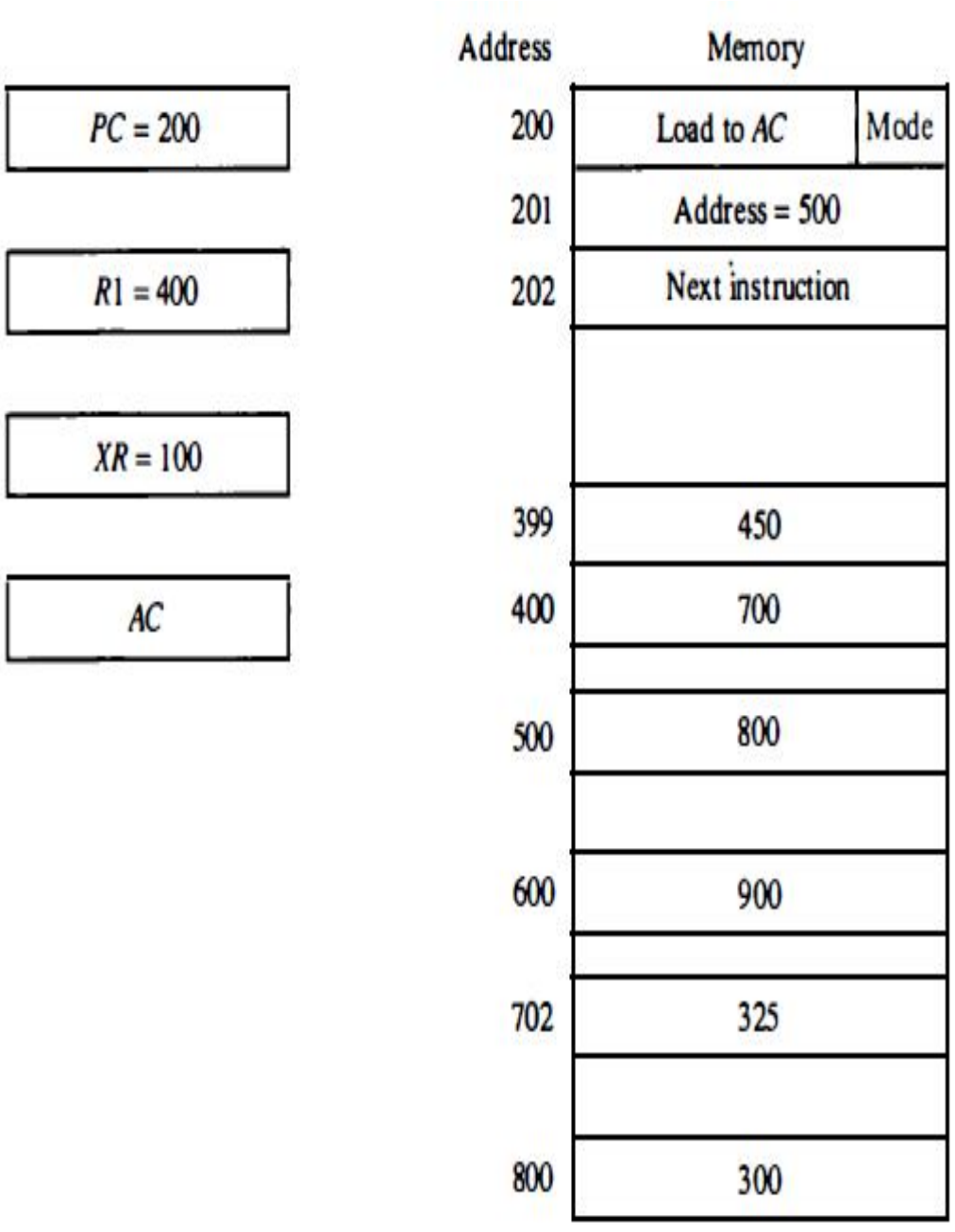


Figure 3: Numerical example for addressing modes

Table 1: Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Description on Addressing Modes

Addressing Mode	Example	Description
Implied / Implicit	CMA, CME, CLA, CLE	Operands are Specified Implicitly in the instruction
Immediate	LDAC #20	Here operands is specified in the instruction itself Here data value 20 is loaded into AC
Direct	LDAC 5	Here 5 refer address of operand. Here instruction reads operand from address location 5 and load into AC
Indirect	LDAC @10 (OR) LDAC (10)	Here first retrieves the content of memory location 10 say 5, then CPU goes to address location 5, read data and load into AC.
Register	LDAC R	Here operands are in Registers. If the register has operand 20, this instruction Loads operand 20 into AC
Register Indirect	LDAC @R LDAC (R)	If the register has address location 5, then CPU goes to

		address location 5, read data from address location 5, and then load into AC
Relative	LADC $5	Assume that Next instruction address is (PC) 3, then add it to the address part of instruction 5 i.e. 5+3→ 8 From the address location 8 the CPU read data and load into AC.
Indexed	LDAC 5(XR)	If the index register XR contains value 10, then this instruction read data from address location 5+10. i.e. address location 15 and load into AC

Control Memory

- The function of the control unit in a digital computer is to initiate sequences of microoperations.
- Two major types of Control Unit.
  i. *Hardwired Control Unit:* The control logic is implemented with gates, Flip-flops, decoders, and other digital circuits, then that control unit is said to be Hardwired Control Unit.
  ii. *Microprogrammed Control Unit:* The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations is referred as Microprogrammed Control Unit.
- *Control Word:* The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- *Microinstruction:* The microinstruction specifies one or more microoperations.
- *Microprogram:* A sequence of microinstruction constitutes a microprogram.
  i. *Static microprogramming:* Here Control Memory = ROM
    - The content of the words in ROM are fixed and cannot be altered by simple programming, since no writing capability is available in the ROM.
    - Control words in ROM are made permanent during the hardware production of the unit.
  ii. *Dynamic microprogramming:* Here Control Memory = RAM
    - Microprogram is 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 for writing (to change the microprogram) but is used mostly for reading.

- A memory that is part of a control unit is referred to as a *control memory*.
- A computer that employs a microprogrammed control unit will have two separate memories: a main memory and a control memory.
- The *main memory* is available to the user for storing the programs. The contents of main memory may alter when the data are manipulated and every time that the program is changed. The user's program in main memory consists of machine instructions and data.
- The *control memory* holds a fixed microprogram that cannot be altered by the occasional user.
- The general configuration of a microprogrammed control unit is demonstrated in the block diagram of the following figure 4.
- The control memory is assumed to be a ROM, within which all control information is permanently stored.

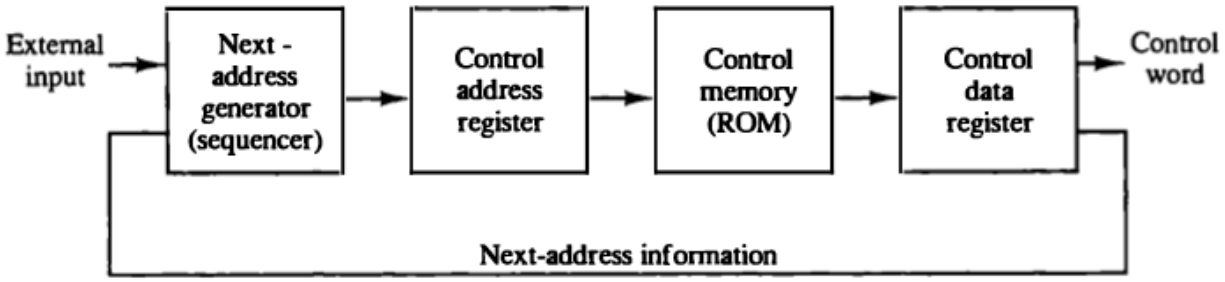


Figure 4: Microprogrammed control organization

- *Control address register:* The control address register specifies the address of the microinstruction.
- *Control data register:* The control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the next address.
- The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory. For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.
- The next address may also be a function of external input conditions. 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 next microinstruction.

- The next address generator is sometimes called a microprogram *sequencer*, as it determines the address sequence that is read from control memory.
- The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs.
- Typical functions of a microprogram 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 is computed and read from memory.
- The data register is sometimes called a *pipeline register*. It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction. This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.
- The other two components: the *sequence*r and the *control memory* are combinational circuits and do not need a clock.
- The main advantage of the microprogrammed 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 control memory.
- The hardware configuration should not be changed for different operations, the only thing that must be changed is the microprogram residing in control memory.

Address Sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a *routine*.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction.
- An initial address is loaded into the control address register when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction *fetch* routine.
- The control memory next must go through the routine that determines the *effective address* of the operand.
- 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.

- The next step is to generate the microoperations that execute the instruction fetched from 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.
- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.
- 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 fetch routine.
- The address sequencing capabilities required in a control memory are:
    1. Incrementing of the control address register.
    2. Unconditional branch or conditional branch, depending on status bit conditions.
    3. A mapping process from the bits of the instruction to an address for control memory.
    4. A facility for subroutine call and return.
- The following figure 5 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.

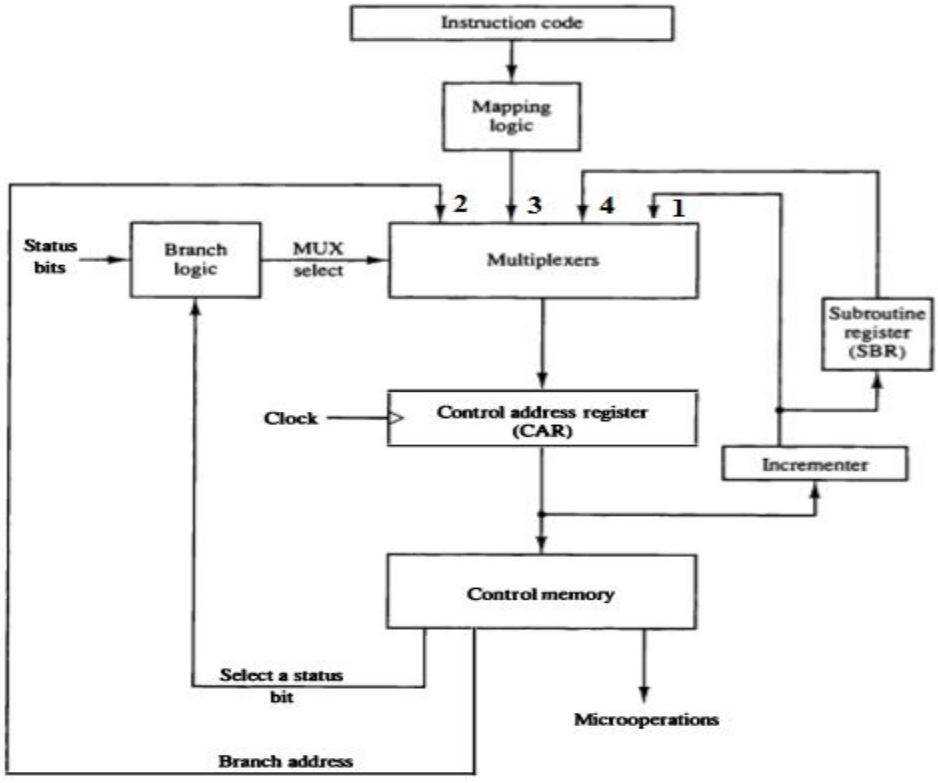


Figure 5: Selection of address for control memory

- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.
- The above diagram shows four different paths from which the control address register (CAR) receives the address.
    1. Incrementer
    2. Branch address from control memory
    3. Mapping Logic
    4. SBR : Subroutine Register
- In the above diagram Multiplexer receives 4 inputs as follows.
    1. CAR Increment
    2. JMP/CALL
    3. Mapping
    4. Subroutine Return
- Subroutine Register (SBR): Return Address cannot be stored in ROM. So return Address for a subroutine is stored in SBR.

Conditional Branching

- The branch logic of figure 5 provides decision-making capabilities in the control unit.
- The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions.
- Information in these bits can be tested and actions initiated based on their condition: whether their value is 1 or 0.
- The Status Bits Control the conditional branch decisions generated in the Branch Logic.
- The Branch LogicTest the specified condition and Branch (jump) to the indicated address if the condition is met; otherwise, the control address register is just incremented.
- An *unconditional branch* microinstruction can be implemented by loading the branch address from control memory into the control address register. This can be accomplished by fixing the value of one status bit at the input of the multiplexer, so it is always equal to 1.

Mapping of 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 is located. 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 6 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. For each operation code there exists a microprogram routine in control memory that executes the instruction.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure 6.
- 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 address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.

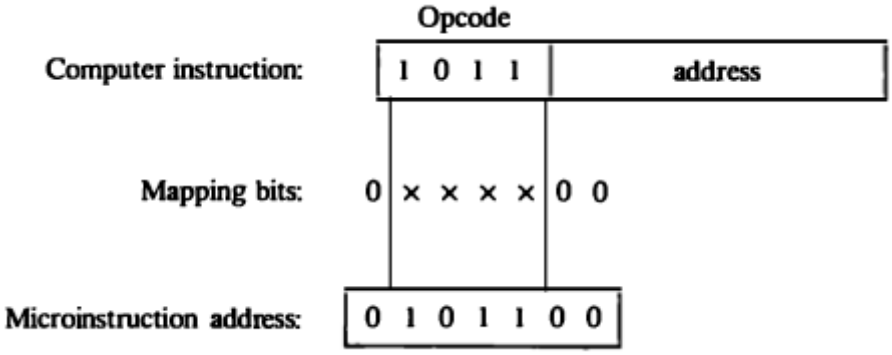


Figure 6: Mapping from instruction code to microinstruction address

Subroutines

- Subroutines are programs that are used by other routines to accomplish a particular task. A subroutine can be called from any point within the main body of the microprogram.
- Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return. This may be accomplished by placing the incremented output from the control address register into a subroutine register and branching to the beginning of the subroutine.
- The subroutine register can then become the source for transferring the address for the return to the main routine.

Microinstruction Format

- The microinstruction format for the control memory is shown in following figure 8. The 20 bits of the microinstruction are divided into four functional parts.

- The three fields F1, F2, and F3 specify microoperations for the computer.
- The CD field selects status bit conditions.
- The BR field specifies the type of branch to be used.
- The AD field contains a branch address.
- The address field is seven bits wide, since the control memory has $128 = 2^7$ words.

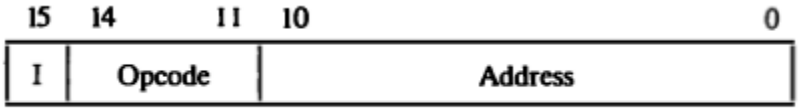
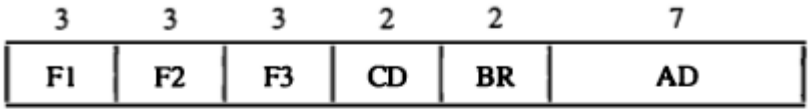


Figure 7: Instruction format

Table 2: Four computer instructions

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Figure 8: Microinstruction code format (20 bits)

- The microoperations are subdivided into three fields of three bits each.

- The three bits in each field are encoded to specify seven distinct microoperations as listed in Table 2. This gives a total of 21 microoperations.
- No more than three microoperations can be chosen for a microinstruction, one from each field. If fewer than three microoperations are used, one or more of the fields will use the binary code 000 for no operation.
- Each microoperation in Table 3 is defined with a register transfer statement and is assigned a symbol for use in a symbolic microprogram.
- All transfer-type microoperations symbols use five letters. The first two letters designate the source register, the third letter is always a *T*, and the last two letters designate the destination register.

Table 3: Symbols and Binary Code for Microinstruction Fields

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

- For example, the microoperation that specifies the transfer AC ←DR (F1 = 100) has the symbol DRTAC, which stands for a transfer from DR to AC.
- The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 4.

Table 4: Symbols and Binary Code for CD field

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*

- The first condition is always a 1, so that a reference to CD = 00 (or the symbol U) will always find the condition to be true. When this condition is used in conjunction with the BR (branch) field, it provides an unconditional branch operation.
- The indirect bit *I* is available from bit 15 of DR after an instruction is read from memory.
- The sign bit of AC provides the next status bit.
- The zero value, symbolized by *Z*, is a binary variable whose value is equal to 1 if all the bits in AC are equal to zero.
- We will use the symbols U, I, S, and Z for the four status bits when we write microprograms in symbolic form.

Table 5: Symbols and Binary Code for BR field

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

- 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.
- As shown in Table 5, when BR = 00, the control performs a jump (JMP) operation (which is similar to a branch), and when BR = 01, it performs a call to subroutine (CALL) operation. The

two operations are identical except that a call microinstruction stores the return address in the subroutine register SBR.

- The jump and call operations depend on the value of the CD field. If the status bit condition specified in the CD field is equal to 1, the next address in the AD field is transferred to the control address register CAR. Otherwise, CAR is incremented by 1.
- The return from subroutine is accomplished with a BR field equal to 10. This causes the transfer of the return address from SBR to CAR.
- The mapping from the operation code bits of the instruction to an address for CAR is accomplished when the BR field is equal to 11. This mapping is as depicted in figure 6.
- The bits of the operation code are in DR(ll-14) after an instruction is read from memory. Note that the last two conditions in the BR field are independent of the values in the CD and AD fields.

Design of Control Unit

- The various fields encountered in instruction formats provide control bits to initiate microoperations in the system, special bits to specify the way that the next address is to be evaluated, and an address field for branching.
- Each field requires a decoder to produce the corresponding control signals.
- The control memory output of each subfield must be decoded to provide the distinct microoperations. The outputs of the decoders are connected to the appropriate inputs in the processor unit.
- The following figure 9 shows the three decoders and some of the connections that must be made from their outputs.

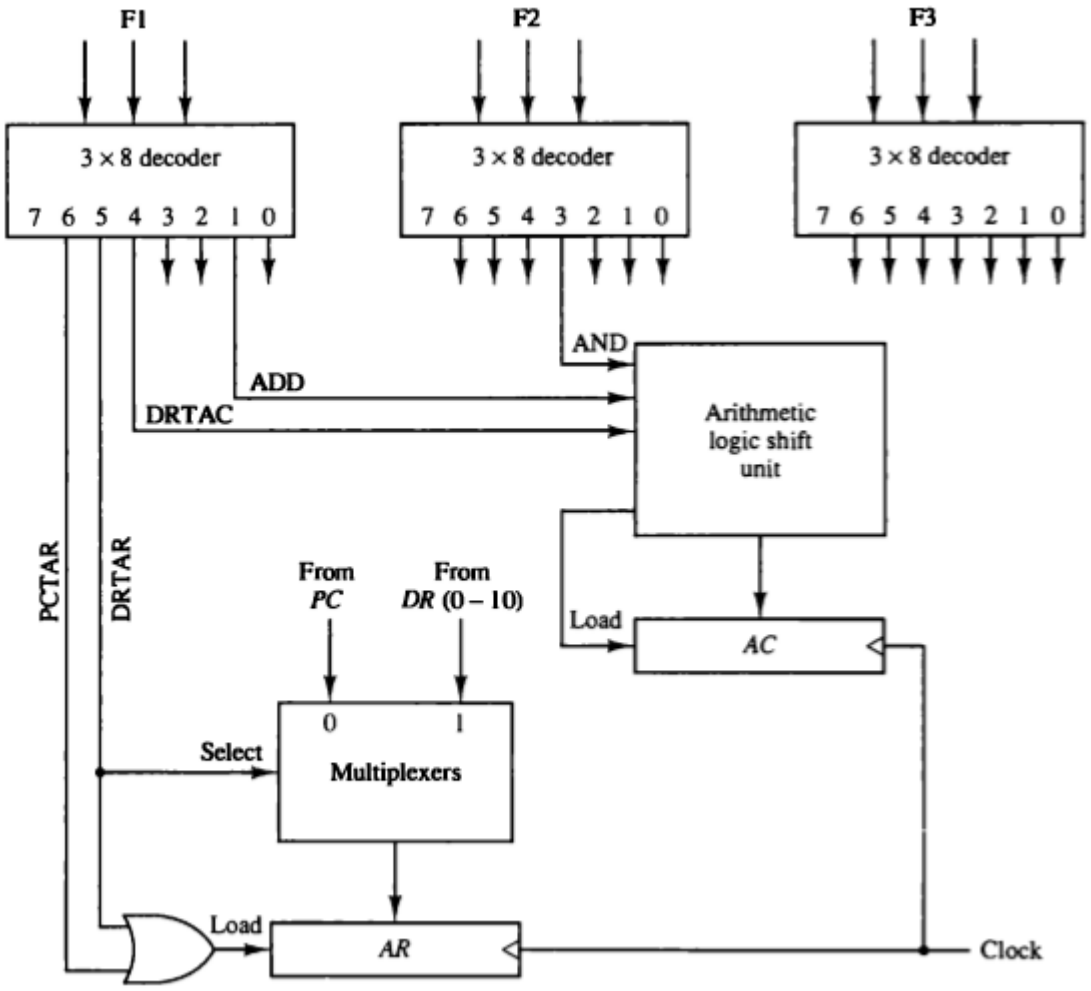


Figure 9: Decoding of microoperation fields

- Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3 x 8 decoder to provide eight outputs.
- Each of these outputs must be connected to the proper circuit to initiate the corresponding microoperation as specified in Table 3.
- For example, when F1 = 101 (binary 5), the next clock pulse transition transfers the content of DR(0-10) to AR (symbolized by DRTAR in Table 3). Similarly, when F1 = 110 (binary 6) there is a transfer from PC toAR (symbolized by PCTAR).
- As shown in figure 9, outputs 5 and 6 of decoder F1 are connected to the load input of AR so that when either one of these outputs is active, information from the multiplexers is transferred to AR.

- The multiplexers select the information from DR when output 5 is active and from PC when output 5 is inactive. The transfer into AR occurs with a clock pulse transition only when output 5 or output 6 of the decoder is active.
- The other outputs of the decoders that initiate transfers between registers must be connected in a similar fashion.
- In arithmetic logic shift unit designing, Instead of using gates to generate the control signals marked by the symbols AND, ADD, and DR, these inputs will now come from the outputs of the decoders associated with the symbols AND, ADD, and DRTAC, respectively, as shown in figure 9.
- The other outputs of the decoders that are associated with an AC operation must also be connected to the arithmetic logic shift unit in a similar fashion.

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 microprogram sequencer.
- A microprogram sequencer can be constructed with digital functions to suit a particular application. However, just as there are large ROM units available in integrated circuit packages, so are general-purpose sequencers suited for the construction of microprogram control units.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next-address logic of the sequencer determines the specific address source to be loaded into the control address register. The choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction.
- The block diagram of the microprogram sequencer is shown in figure 10. The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.

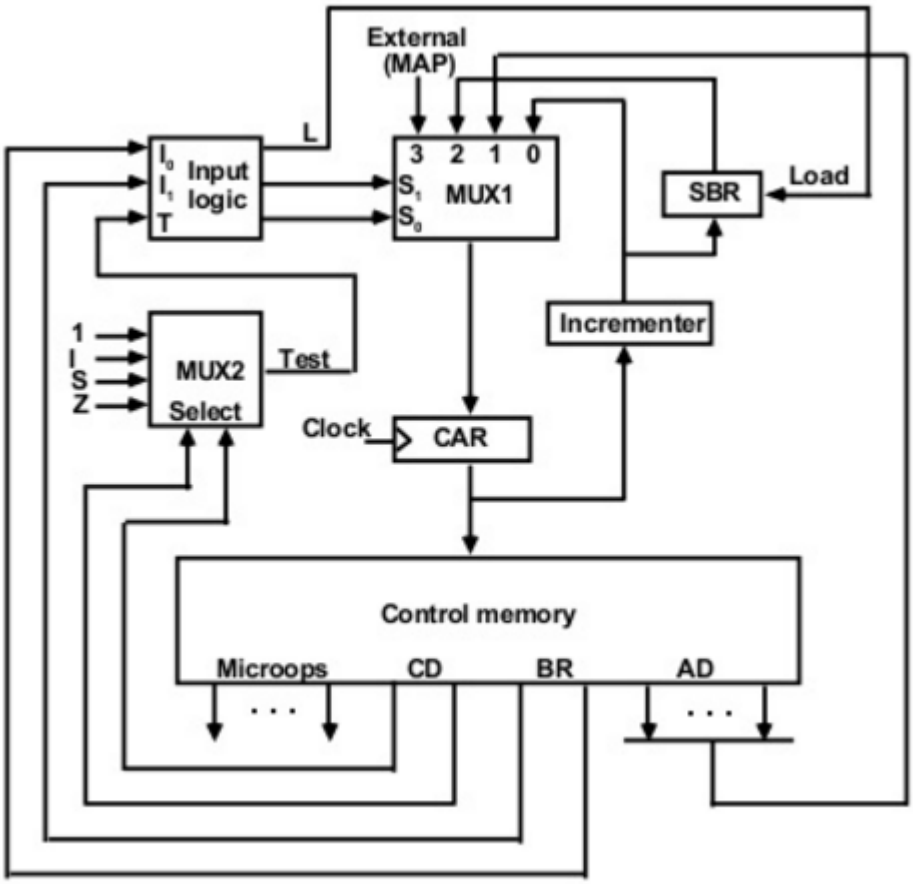


Figure 10: Microprogram sequencer for a control memory.

- There are two multiplexers in the circuit. The first multiplexer selects an address from one of four sources and routes it into a control address register CAR. 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 control memory. The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register SBR.
- The other three inputs to multiplexer number 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction.
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer. If the bit selected is equal to 1, the T (test) variable is equal to 1, otherwise it is equal to 0.
- The T value together with the two bits from the BR (branch) field go to an input logic circuit. The input logic in a particular sequencer will determine the type of operations that are available in the unit.

- The input logic circuit in figure 10 has three inputs, I_0, I_1, and T, and three outputs, S_0, S_1, and L.
- Variables S_0 and S_1, select one of the source addresses for CAR.
- Variable L enables the load input in SBR.
- The binary values of the two selection variables determine the path in the multiplexer. For example, with S_1 S_0 = 10, multiplexer input number 2 is selected and establishes a transfer path from SBR to CAR.
- The truth table for the input logic circuit is shown in Table 6. Inputs I_1 and I_0 are Identical to the bit values in the BR field.
- The function listed in each entry was defined in Table 3. The bit values for S_1 and S_0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.
- The subroutine register is loaded with the incremented value of CAR during a call microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1).
- The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I_1' T$$
$$L = I_1' I_0 T$$

Table 6: Input logic Truth Table for MicroProgram Sequence

BR Field	Input I_1 I_0 T	MUX 1 S_1 S_0	Load *SBR* 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 ×	1 0	0
1 1	1 1 ×	1 1	0

Hardwired control unit

- In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation.

- A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.
- A hardwired control for the basic computer is as follows. The block diagram of the control unit is shown in following figure 11.

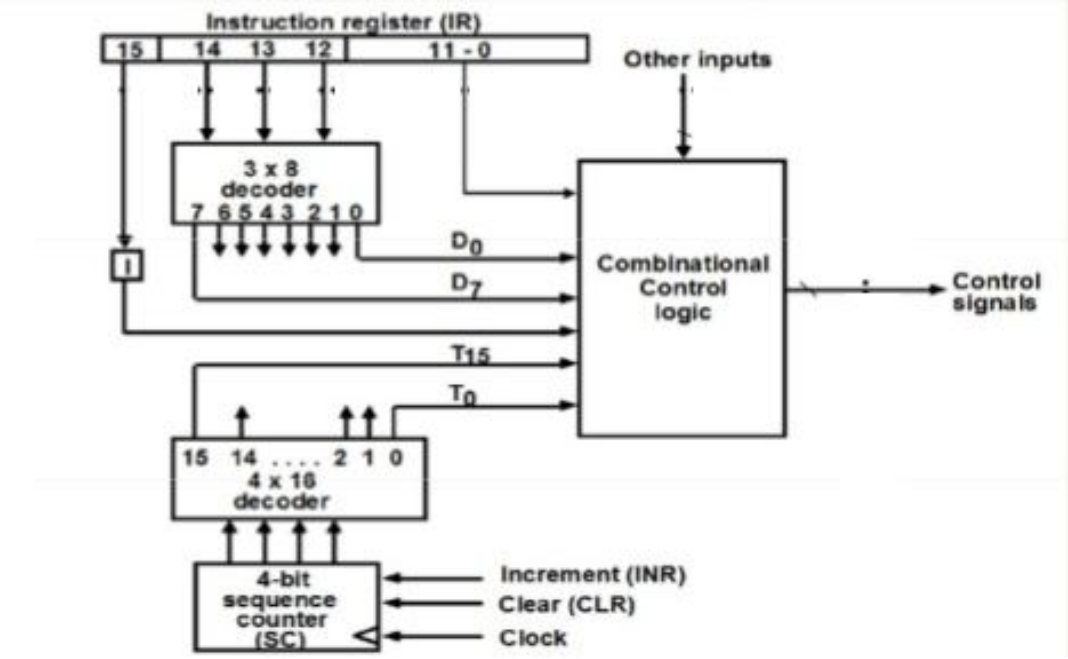


Figure 11: Control unit of basic computer

- It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR).
- The instruction register is shown in figure 11, where it is divided into three parts: the *I* bit, the operation code, and bits 0 through 11.
- 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 through D_7.
- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.
- The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder.

- Once in awhile, the counter is cleared to 0, causing the next active timing signal to be T_0. 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 D_3 is active.

Hardwired control unit	Micro-programmed Control Unit
1. It uses flags, decoder, logic gates and other digital circuits.	1. It uses sequence of micro-instruction in micro programming language.
2. As name implies it is a hardware control unit.	2. It is mid-way between Hardware and Software.
3. On the basis of input Signal output is generated.	3.It generates a set of control signal on the basis of control line.
4. Difficult to design, test and implement.	4. Easy to design, test and implement.
5. Inflexible to modify.	5. Flexible to modify.
6. Faster mode of operation.	6. Slower mode of operation.
7. Expensive and high error.	7. Cheaper and less error.
8. Used in RISC processor	8. Used in CISC processor.

Unit - II

Assignment-Cum-Tutorial Questions

Section – A

1. The instruction ADD R1, 30FF is belongs to _____ [ ]

   a) A 3-address instruction format b) A 2-address instruction format

   c) A 1-address instruction format d) A 0-address instruction format

2. Which of the following options represents the correct matching? [ ]

Addressing Mode	Description
1. Immediate	A. The address field contains the address (in main memory) where the operand is stored
2. Direct	B. The address field refers to the address of a word in the memory, which in-turn contains the address of the operand
3. Indirect	C. The address field of the operand is a register
4. Register Direct	D. Operand value is present in the instruction itself (address field)

   a) 1→A; 2→D; 3→C; 4→B; b) 1→D; 2→A; 3→B; 4→C;

   c) 1→D; 2→A; 3→C; 4→B; d) 1→A; 2→D; 3→B; 4→C;

3. The instruction, Add #45,R1 does _____ [ ]

   a) Adds the value of 45 to the address of R1 and stores 45 in that address

   b) Adds 45 to the value of R1 and stores it in R1

   c) Finds the memory location 45 and adds that content to that of R1

   d) None of the mentioned

4. The instruction, MOV AX, 0005H belongs to the address mode_____ [ ]

   a) Register b) Direct c) Immediate d)Register relative

5. The addressing mode used in an instruction of the form ADD X, Y is_____? [ ]

   a) Direct b) Immediate c) Indirect d) Register

6. The addressing mode used in the instruction PUSH B is_____? [ ]

   a) Direct b) Register c) Register indirect d) Index

7. The addressing mode, where you directly specify the operand value is ___. [ ]

a) Immediate b) Direct c) Definite d) Relative

8. The addressing mode which makes use of in-direction pointers is _____. [ ]

a) Indirect b) Index c) Relative d) Offset

9. A sequence of microinstructions constitutes a_____ [ ]

a) microprogram b) microoperation

c) microinstruction d) microprocessor

10. A memory that is part of control unit is referred to as _____ [ ]

a) cache memory b) control memory

c) main memory d) virtual memory

11. When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be_____ [ ]

a) programmed b) microprogrammed

c) hardwired d) none of the above

12. The next address generator is also called_____ [ ]

a) microprogram sequencer b) control unit

c) micro instruction sequencer d) microprogrammed control

13. The goals of both hardwired control and microprogrammed control units are to_____

a) access memory b) access ALU [ ]

c) cost a lot of money d) generate control signals

14. Each computer instruction has its own _____in control memory to generate the microoperations that execute the instruction. [ ]

a) microinstruction b) branch instruction

c) mapping logic d) microprogram routine

15. The control_____ register specifies the address of the microinstruction, and the control _____register holds the microinstruction read from memory. [ ]

a) Address, data b) data, memory

c) memory, instruction d) data , instruction

16. A control memory has 4096 words of 24 bits each. How many bits are there in the control address register? [ ]

a) 12 b) 24 c) 32 d) 25

17. The branch logic that provides decision making capabilities in the control unit is known as_____ [ ]

a) controlled branching b) conditional branching

c) unconditional branching d) none of the above

18. The microprogram written as strings of 0's and 1's is a_____ [ ]

a) symbolic microinstruction b) binary microinstruction

c) symbolic microprogram d) binary microprogram

19. Address information in the microinstruction ***cannot*** be_____ [ ]

a) single address field b) two address field

c) variable format d) a control signal

20. Micro instruction execution is_____ [ ]

a) To generate the control signals needed to execute the microinstruction

b) To get the next microinstruction from the control memory

c) To get the next instruction from the main memory

d) To get the microinstruction from the main memory

21. Microinstruction sequencing is_____ [ ]

a) to get the next instruction from the main memory

b) to get the next microinstruction from the control memory

c) to get the microinstruction from the main memory

d) to generate the control signals needed to execute the micro instruction

22. A microprogram sequencer

a) generates the address of next micro instruction to be executed

b) generates the control signals to execute a microinstruction.

c) sequentially averages all microinstructions in the control memory.

d) enables the efficient handling of a micro program subroutine.

Section – B

1. Illustrate different types of Instruction formats.
2. Describe different types of Addressing modes with examples.
3. Write short notes on control memory.
4. Explain the process of mapping with an example.

5. Explain the use of subroutine register.
6. Explain how control signals are generated using hard wired control signals.
7. Explain how control signals are generated using micro programmed control signals.
8. Define the following: Micro operations, Micro instructions, Micro program, Micro code.
9. Differentiate between hardwired control unit and micro programmed control unit.
10. What is Microprogramming? Explain with a simple example.
11. Explain how the next instruction address is generated.
12. Using the mapping procedure generate the micro instruction address for the following op-codes.
    a. 0010
    b. 1011
    c. 1111
13. What are elements required in designing a Control Unit.
14. Few bits of the current micro instruction are used to generate the address of the next micro instruction to be executed. Explain why?
15. Using the mapping procedure give the first micro instruction address for the following op-code.
    a. 0101
    b. 1010
    c. 0001

Section - C

1. In the absolute addressing mode: **(GATE-CS-2002) [ ]**
    a) the operand is inside the instruction
    b) the address of the operand is inside the instruction
    c) the register containing address of the operand is specified inside the instruction
    d) the location of the operand is implicit

2. Which is the most appropriate match for the items in the first column with the items in the second column: **(GATE-CS-2001) [ ]**

X. Indirect Addressing	I. Array implementation
Y. Indexed Addressing	II. Writing re-locatable code
Z. Base Register Addressing	III. Passing array as parameter

a) (X, III) (Y, I) (Z, II) b) (X, II) (Y, III) (Z, I)

c) (X, III) (Y, II) (Z, I) d) (X, I) (Y, III) (Z, II)

3. The most appropriate matching for the following pairs: **(GATE-CS-2000) [ ]**

X: Indirect addressing	1: Loops
Y: Immediate addressing	2: Pointers
Z: Auto decrement addressing	3: Constants

a) X-3, Y-2, Z-1 b) X-I, Y-3, Z-2

c) X-2, Y-3, Z-1 d) X-3, Y-l, Z-2

4. Which of the following addressing modes are suitable for program relocation at run time?

(i) Absolute addressing (ii) Based addressing **(GATE-CS-2004)**

(iii) Relative addressing (iv) Indirect addressing

a) (i) and (iv) b) (i) and (ii) c) (ii) and (iii) d) (i), (ii) and (iv)

5. Identify the addressing modes of below instructions and match them: [ ]

(Paper-3 NET June 2012)

(A) ADI	(1) Immediate addressing
(B) STA	(2) Direct addressing
(C) CMA	(3) Implied addressing
(D) SUB	(4) Register addressing

a) A – 1, B – 2, C – 3, D – 4 b) A – 2, B – 1, C – 4, D – 3

c) A – 3, B – 2, C – 1, D – 4 d) A – 4, B – 3, C – 2, D – 1

6. Consider a hypothetical processor with an instruction of type LW R1, 20(R2), which during execution reads a 32-bit word from memory and stores it in a 32-bit register R1. The effective address of the memory location is obtained by the addition of a constant 20 and the contents of register R2. Which of the following best reflects the addressing mode implemented by this instruction for the operand in memory? **(GATE 2011) [ ]**

a) Immediate Addressing b) Register Addressing

c) Register Indirect Scaled Addressing d) Base Indexed Addressing

7. A microprogrammed control unit_____ (**GATE 1987**) [ ]

a) Is faster than hardwired control unit

b) Facilitates easy implementation of new instructions

c) Is useful when very small programs are run

d) Usually refers to control unit of microprocessor

8. The effective address of MUL 5(R1,R2) instruction is_____ [ ]

a) 5+R1+R2 b) 5+(R1*R2) c) 5+[R1]+[R2]. d) 5*([R1]+[R2])

9. Match each of the high level language statements given on the left hand side with the most natural addressing mode from those listed on the right hand side. [ ]

1. A[1] = B[J];	A) Indirect addressing
2. while [*A++];	B) Indexed addressing
3. int temp = *x;	C) Autoincrement

a) (1, C), (2, B), (3, A) b) (1, A), (2, C), (3, B)

c) (1, B), (2, C), (3, A) d) (1, A), (2, B), (3, C)

UNIT – III

MEMORY ORGANIZATION

Objective:

- To familiarize with organizational aspects of memory.

Syllabus:

MEMORY ORGANIZATION: Memory Hierarchy, main memory, auxiliary memory, associative memory, cache memory, virtual memory.

Learning Outcomes:

At the end of the unit student will be able to:

- Differentiate different types of memories.
- Analyze the performance of the hierarchical organization of memory.

Learning Material

3.1 MEMORY HIERARCHY

- The memory hierarchy system is shown in the following figure 1.

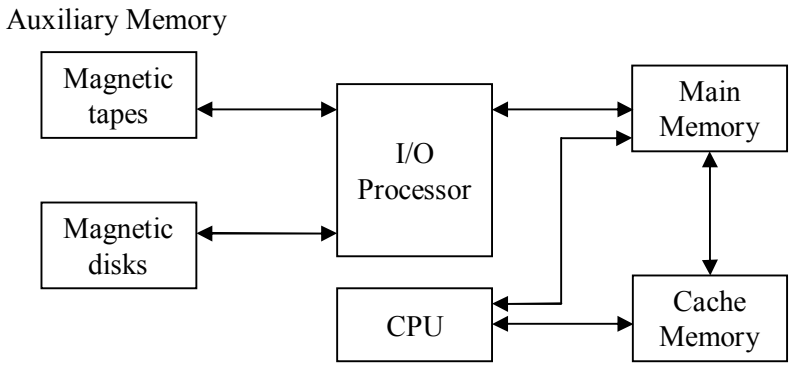


Figure 1: Memory Hierarchy in computer system

- The memory unit that communicates directly with the CPU is called the *main memory*.
- Devices that provide backup storage are called *auxiliary memory*. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backup information.
- Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.

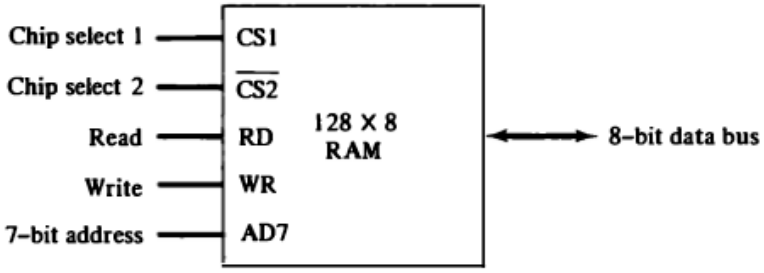
- A special very high speed memory called a *Cache* is used to increase the speed of processing by making current programs and data available to the CPU.
- The part of the computer system that supervises the flow of information between auxiliary memory and main memory is called the *memory management system*.

3.2 MAIN MEMORY

- The main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation.
- The principal technology used for the main memory is based on semiconductor integrated circuits such as RAM and ROM chips.
- RAM chips are available in two possible operating modes, static and dynamic.
- The static RAM consists of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit.
- The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors.
- The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles.
- Most of the main memory in a general-purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips.
- RAM is used for storing the bulk of the programs and data that are subject to change.
- ROM is used for storing programs that are permanently resident in the computer.
- The ROM portion of main memory is needed for storing an initial program called a *bootstrap loader*.
- The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.
- Since RAM is *volatile*, its contents are destroyed when power is turned off. The contents of ROM remain unchanged after power is turned off and on again.
- RAM and ROM chips are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size.

3.2.1 RAM Chip

- A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed.
- Another common feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a *read* operation or from CPU to memory during a *write* operation.
- The block diagram of a RAM chip is shown in figure2. The capacity of the memory is 128 words of eight bits (one byte) per word.



(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

Figure 2: Typical RAM chip

- It requires a 7-bit address bus and an 8-bit bidirectional data bus.
- The read and write inputs specifies the memory operation and the two chips select (CS) control inputs are for enabling the chip only when it is selected by the microprocessor.
- The availability of more than one control input to select the chip facilitates the decoding the address lines when multiple chips are used in the microcomputer.
- The function table listed in figure 2(b) specifies the operation of the RAM chip.
- The unit in operation only when CS1 = 1 and $\overline{CS2}$ = 0.
- If the chip select inputs are not enabled, or if chip select inputs are enabled but the READ or WRITE inputs are not enabled, then the memory in Inhibit and its data bus is in high-impedance state.

3.2.2 ROM Chip

- A ROM chip is organized externally in a similar manner of RAM chip.
- The block diagram of a ROM chip is shown in Figure3.
- The nine address lines in the ROM chip specify any one of the 512 bytes stored in it.
- The two chip select inputs must be CS1 = 1 and $\overline{CS2}$ = 0 for the unit to operate. Otherwise, the data bus is in a high-impedance state.
- There is no need for a read or write control because the unit is read only.

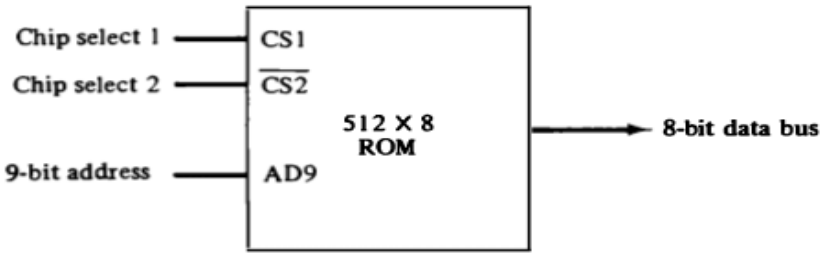


Figure 3: Typical ROM chip

3.2.3 Memory Address Map

- The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM.
- The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available.
- The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a memory address map, is a pictorial representation of assigned address space for each chip in the system.
- The memory address map for this configuration is shown in Table 1. The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column.

Table 1: Memory Address Map for Microcomputer

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

- Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero.

3.2.4 Memory Connection to CPU

- RAM and ROM chips are connected to a CPU through the data and address buses. The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs. The connection of memory chips to the CPU is shown in Figure 4.
- This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM.
- Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determined from lines 8 and 9 in the address bus. This is done through a 2 x 4 decoder whose outputs go to the *CS1* inputs in each RAM chip.
- Thus, when address lines 8 and 9 are equal to 00, the first RAM chip is selected. When 01, the second RAM chip is selected, and so on.
- The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip.
- The selection between RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0, and the ROM when the bit is 1.

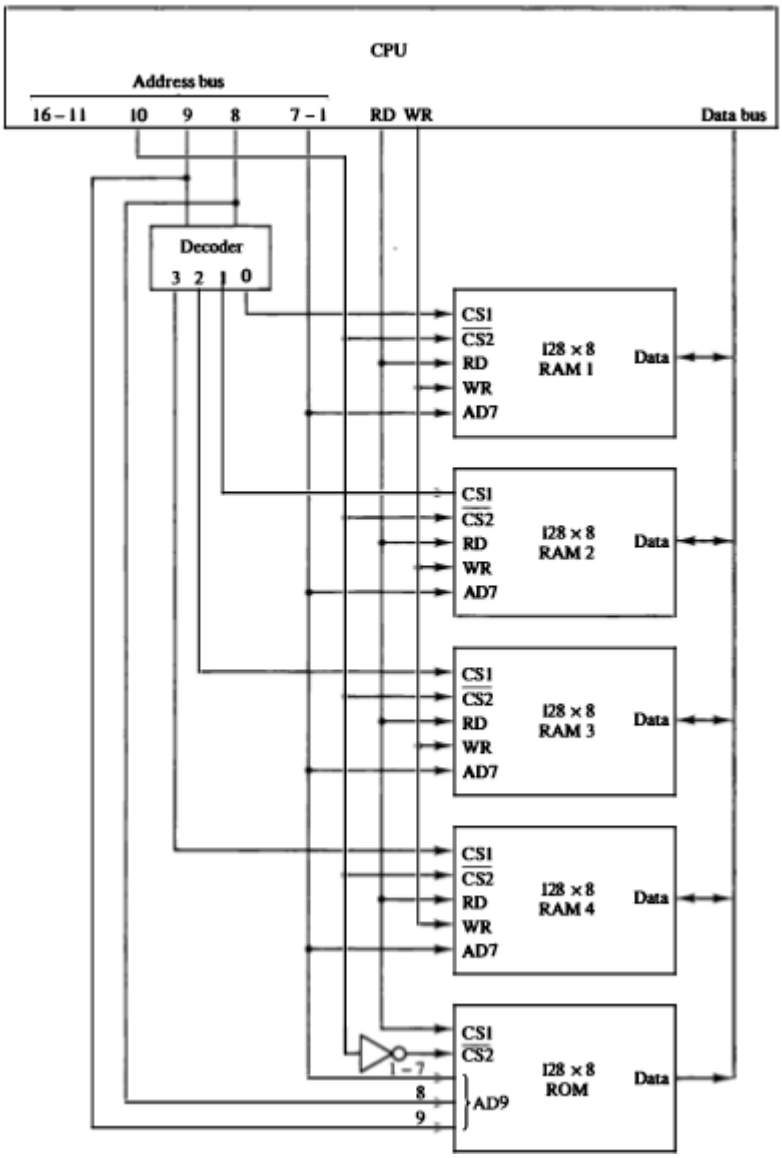


Figure 4: Memory connection to the CPU

3.3 AUXILIARY MEMORY

- The most common auxiliary memory devices used in computer systems are magnetic disks and tapes.
- The average time required to reach a storage location in memory and obtain its contents is called the *access time*.
- In electromechanical devices with moving parts such as disks and tapes, the access time consists of a *seek time* required to position the read-write head to a location and a transfer time required to transfer data to or from the device.

- Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a *write head*. Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a *read head*.

3.3.1 Magnetic Disks

- A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface.
- All disks rotate together at high speed and are not stopped or started for access purposes. Bits are stored in the magnetized surface in spots along concentric circles called *tracks*.
- The tracks are commonly divided into sections called *sectors*. In most systems, the minimum quantity of information which can be transferred is a sector. The subdivision of one disk surface into tracks and sectors is shown in Figure 5.

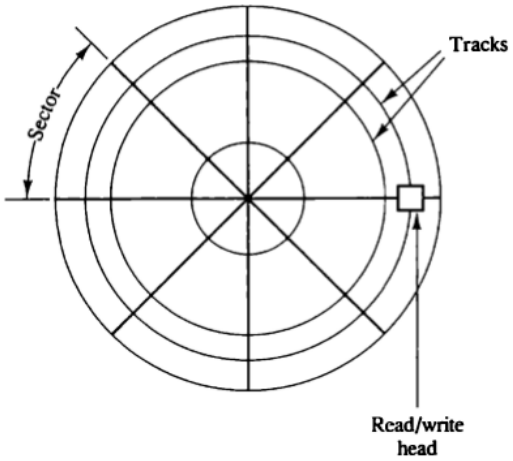


Figure 5: Magnetic disk

- Disks that are permanently attached to the unit assembly and cannot be removed by the occasional user are called *hard disks*. A disk drive with removable disks is called a *floppy disk*.

3.3.2 Magnetic Tape

- A magnetic tape transport consists of the electrical, mechanical, and electronic components to provide the parts and control mechanism for a magnetic-tape unit.
- The tape itself is a strip of plastic coated with a magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks. Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit.
- Read/write heads are mounted one in each track so that data can be recorded and read as a sequence of characters.

- Magnetic tape units can be stopped, started to move forward or in reverse, or can be rewind.

3.4 ASSOCIATIVE MEMORY

- The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- A memory unit accessed by content is called an associative memory or content addressable memory (CAM).
- This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.
- When a word is to be read from an associative memory, the content of the word, or part of the word, is specified. The memory locates all words which match the specified content and marks them for reading. So the associative memory is uniquely suited to do parallel searches by data association.
- An associative memory is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument. For this reason, associative memories are used in applications where the search time is very critical and must be very short.

3.4.1 Hardware Organization

- The block diagram of an associative memory is shown in Figure 6. It consists of a memory array and logic for *m* words with *n* bits per word.

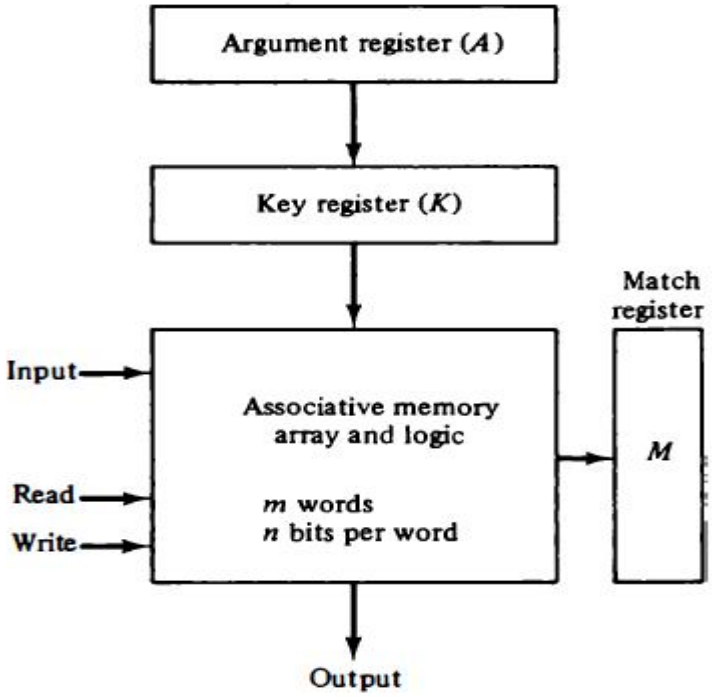


Figure 6: Block diagram of associative memory

- The argument register A and key register K each have *n* bits, one for each bit of a word. The match register M has *m* bits, one for each memory word.
- Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register.
- After the matching process, those bits in the match register that have been set indicate that their corresponding words have been matched.
- Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.
- The key register provides a mask for choosing a particular field or key in the argument word.
- The entire argument is compared with each memory word if the key register contains all l's. Otherwise, only those bits in the argument that have l's in their corresponding position of the key register are compared. Thus the key provides a mask or identifying piece of information which specifies how the reference to memory is made.
- For example the argument registers A and the key register K have the bit configuration shown below. Only the three leftmost bits of A are compared with memory words because K has 1's in these positions.

A 101 111100

K	111 000000	
Word 1	100 111100	*no match*
Word 2	101 000001	*match*

- Word 2 matches the unmasked argument field because the three leftmost bits of the argument and the word are equal.
- The relation between the memory array and external registers in an associative memory is shown in Figure 7.
- The cells in the array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. Thus cell C_{ij} is the cell for bit j in word i.
- Bit A_j in the argument register is compared with all the bits in column j of the array provided that $K_j = 1$.
- This is done for all columns j = 1, 2, . . . , n. If a match occurs between all the unmasked bits of the argument and the bits in word i, the corresponding bit M_i in the match register is set to 1.
- If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.

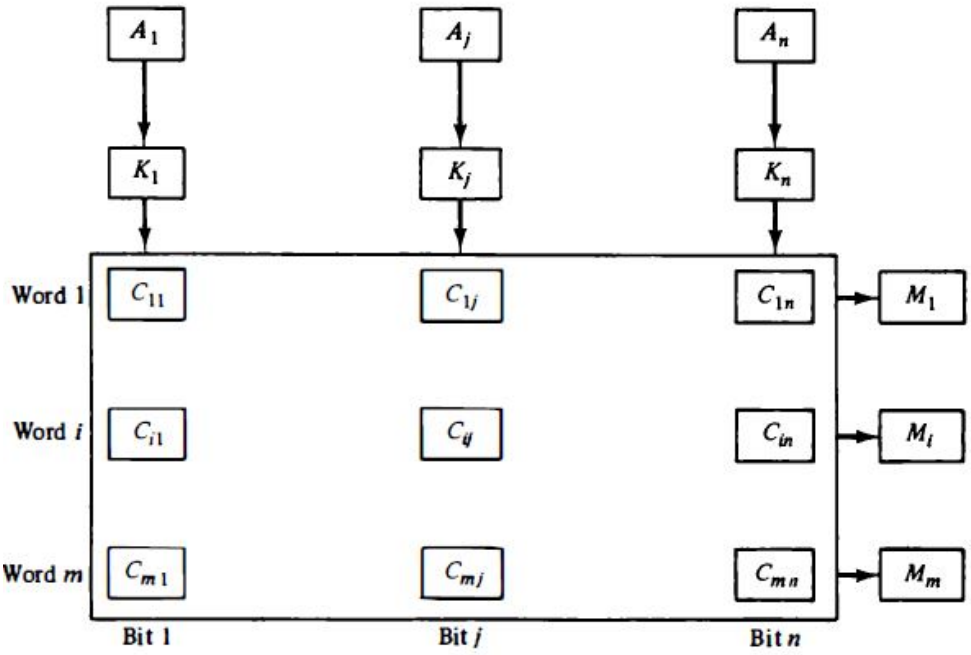


Figure 7: Associative memory of m word, n cells per word

- The internal organization of a typical cell C_{ij} is shown in Figure 8. It consists of a flip-flop storage element F_{ij} and the circuits for reading, writing, and matching the cell.

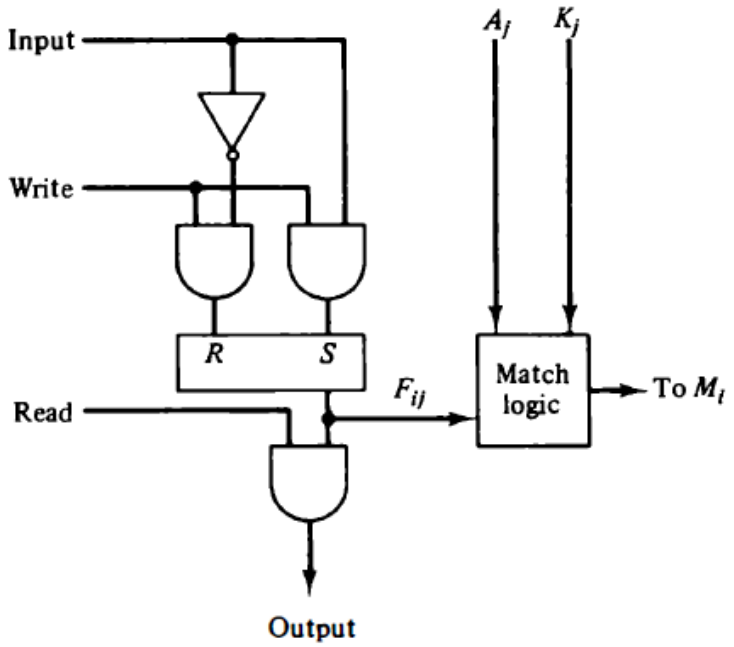


Figure 8: One cell of associative memory

- The input bit is transferred into the storage cell during a write operation. The bit stored is read out during a read operation.
- The match logic compares the content of the storage cell with the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in M_i.

3.4.2 Match Logic

- The match logic for each word can be derived from the comparison algorithm for two binary numbers. First, we *neglect* the key bits and compare the argument in A with the bits stored in the cells of the words.
- Word i is equal to the argument in A if $A_i = F_{ij}$ for j = 1, 2, . . . , n . Two bits are equal if they are both 1 or both 0. The equality of two bits can be expressed logically by the Boolean function

$$x_j = A_j F_{ij} + A'_j F'_{ij}$$

- Where $x_i = 1$ if the pair of bits in position j are equal; otherwise, $x_i = 0$.
- For a word i to be equal to the argument in A we must have all x_j variables equal to 1. This is the condition for setting the corresponding match bit M_i to 1. The Boolean function for this condition is

$$M_i = x_1 x_2 x_3 \ldots . x_n$$

and constitutes the AND operation of all pairs of matched bits in a word.

- Now include the key bit K_j in the comparison logic. The requirement is that if $K_j = 0$, the corresponding bits of A_j and f_{ij} need no comparison. Only when $K_j = 1$ must they be compared. This requirement is achieved by ORing each term with K_j' thus

$$x_j + K_j' = \begin{cases} x_j & \text{if } K_j = 1 \\ 1 & \text{if } K_j = 0 \end{cases}$$

- The match logic for word i in an associative memory can now be expressed by the following Boolean function

$$M_i = (x_1 + K_1')(x_2 + K_2')(x_3 + K_3') \cdots (x_n + K_n')$$

- The circuit for matching one word is shown in figure 9. Each cell requires two AND gates and one OR gate.

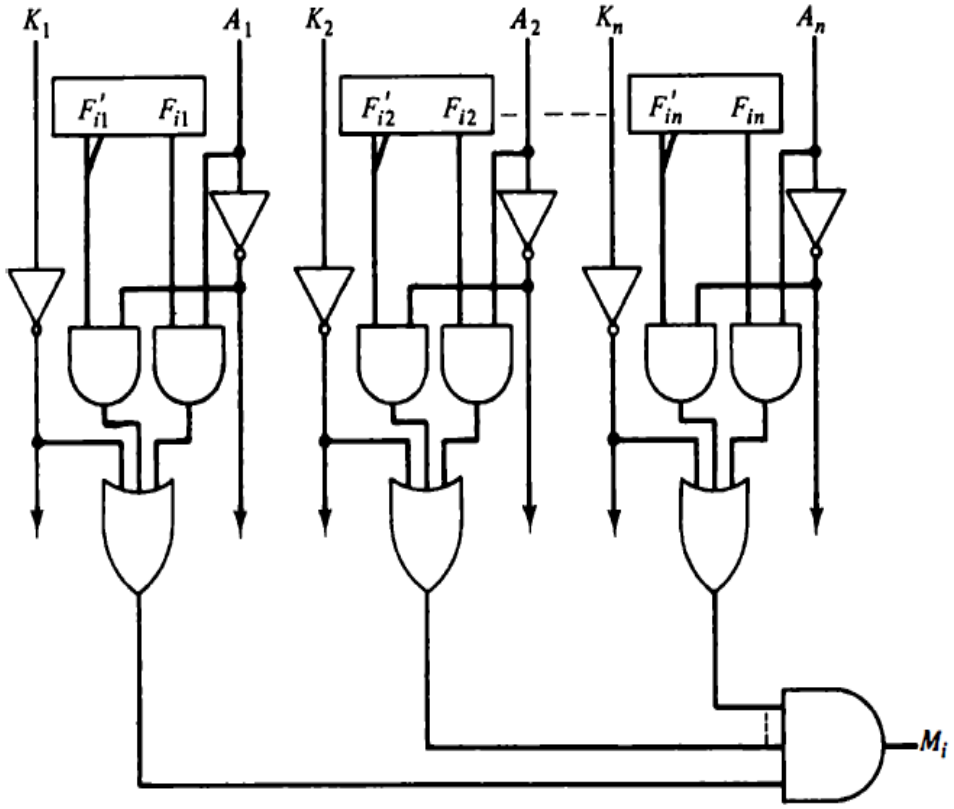


Figure 9: Match logic for one word of associative memory

- The inverters for A_j and K_j are needed once for each column and are used for all bits in the column. The output of all OR gates in the cells of the same word go to the input of a common AND gate to generate the match signal for M_i.
- M_i will be logic 1 if a match occurs and 0 if no match occurs.

3.4.3 Read Operation

- If more than one word in memory matches the unmasked argument field, all the matched words will have 1's in the corresponding bit position of the match register. It is then necessary to scan the bits of the match register one at a time.
- The matched words are read in sequence by applying a read signal to each word line whose corresponding M_i bit is a 1.
- In most applications, the associative memory stores a table with no two identical items under a given key. In this case, only one word may match the unmasked argument field.

3.4.4 Write Operation

- Writing in an associative memory can take different forms, depending on the application.
- If the entire memory is loaded with new information at once prior to a search operation then the writing can be done by addressing each location in sequence. This will make the device a random access memory for writing and a content addressable memory for reading.
- The advantage here is that the address for input can be decoded as in a random access memory. Thus instead of having m address lines, one for each word in memory, the number of address lines can be reduced by the decoder to d lines, where $m = 2^d$.

3.5 CACHE MEMORY

- Analysis of a large number of typical programs has shown that the references to memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of *locality of reference*.
- If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a cache memory.
- Cache memory is placed between the CPU and main memory as illustrated in Figure 1. The cache memory access time is less than the access time of main memory by a factor of 5 to 10.
- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.
- The fundamental idea of cache organization is to reduce the speed mismatch between CPU and main memory.
- The basic operation of the cache is, when the CPU needs to access memory, the cache is examined. If the required data is found in the cache, it is read from it.

- If the required data by the CPU is not found in the cache, then the main memory is accessed to read the required data, just accessed data is then transferred from main memory to cache memory.
- The performance of cache memory is frequently measured in terms of a quantity called hit ratio.
- When the CPU refers to memory and finds the required data in cache, it is said to produce a *hit*. If the required data is not found in cache, it is in main memory and it counts as a *miss*.

$$Hit\ ratio = \frac{number\ of\ hits}{number\ of\ hits + number\ of\ miss}$$

- The transformation of data from main memory to cache memory is referred to as a *mapping* process.
- Three types of *mapping techniques* for cache memory as follows.
    1. Associative mapping
    2. Direct mapping
    3. Set-associative mapping
- For these three mapping procedures one example of a memory organization as shown in following Figure 10.
- The main memory can store 32K words of 12 bits each. i.e. 32K×12 of main memory.
- The cache is capable of storing 512 of these words of 12 bits each. i.e 512×12 of cache memory.

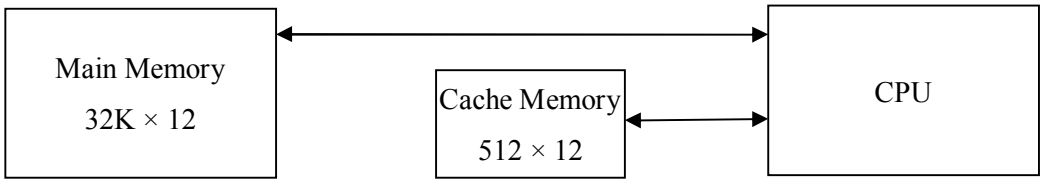


Figure 10: Example of cache memory

- For every word stored in cache, there is duplicate copy in main memory.

3.5.1 Associative Mapping

- The fastest and most flexible cache organization uses an associative memory. This organization is illustrated in Figure 11.
- The associative memory stores both the address and content (data) of the memory word.
- For example Main Memory → 32K×12 → 2^{15}× 12 ➔ 15 Address lines and 12 data lines.
- Cache Memory→ 512 × 12 → 2^{9}× 12 ➔ 9 Address lines and 12 data lines.
- The address value of 15 bits is shown as a five-digit octal number and its corresponding 12 -bit word is shown as a four-digit octal number.

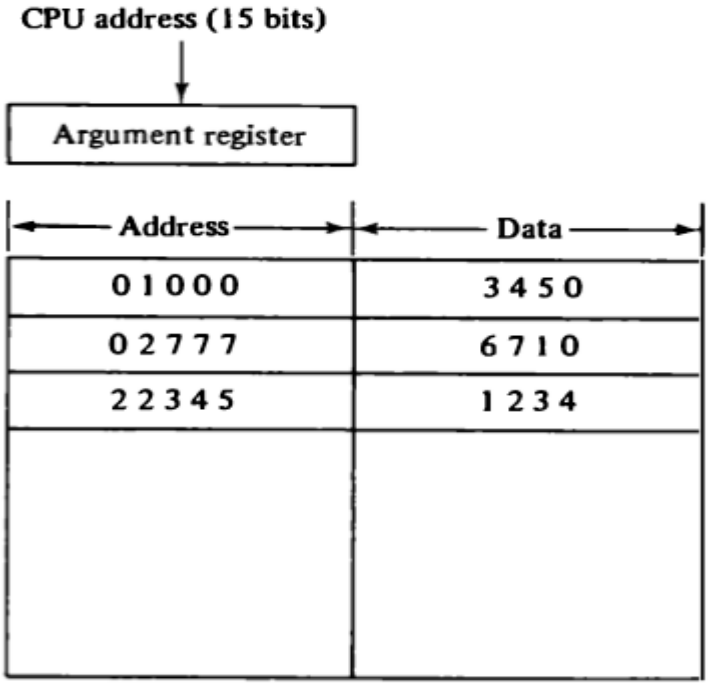


Figure 11: Associative mapping cache (all numbers in octal)

- A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.
- If the address is found, the corresponding 12-bit data is read and sent to the CPU.
- If the address is not matched, then CPU accesses the main memory for the data. The address and data pair is then transferred to the associative cache memory.
- If the cache is full then replacement is occur in cache memory.

3.5.2 Direct Mapping

- Associative memories are expensive compared to random-access memories because of the added logic associated with each cell.
- Here the CPU address bits are divided into two fields named as *tag* field and *index* field.
- The number of bits in the index field is equal to the number of address bits required to access the cache memory.
- The number of bits for tag field is the number of bits required to represent the address of Main Memory - number of bits required to represent the address of Cache Memory.
- For example: MM → 32K × 12 ➔ 15 *address lines* and 12 *data lines*
  CM → 512 × 12 ➔ 9 *address lines* and 12 *data lines*

- So CPU address of 15 bits is divided into two fields. The 9 least significant bits constitute the *index* field and the remaining 6 bits form the *tag* field.
- In the general case, there are 2^k words in cache memory and 2^n words in main memory. The n-bit memory address is divided into two fields as *k* bits for the *index* field and *n* - *k* bits for the *tag* field.
- The direct mapping cache organization uses the n-bit address to access the main memory and the k-bit index to access the cache.

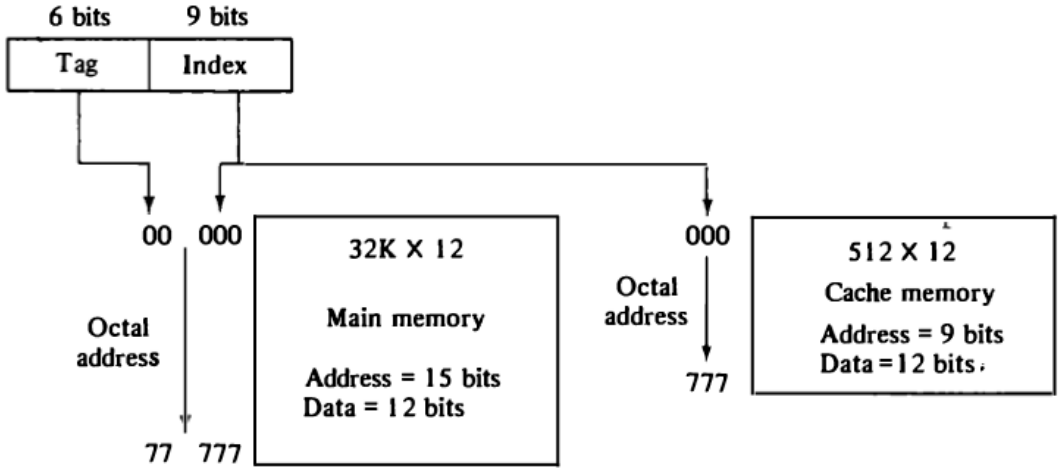


Figure 12: Addressing relationships between main and cache memories

- The internal organization of the words in the cache memory is as shown in figure 13(b).

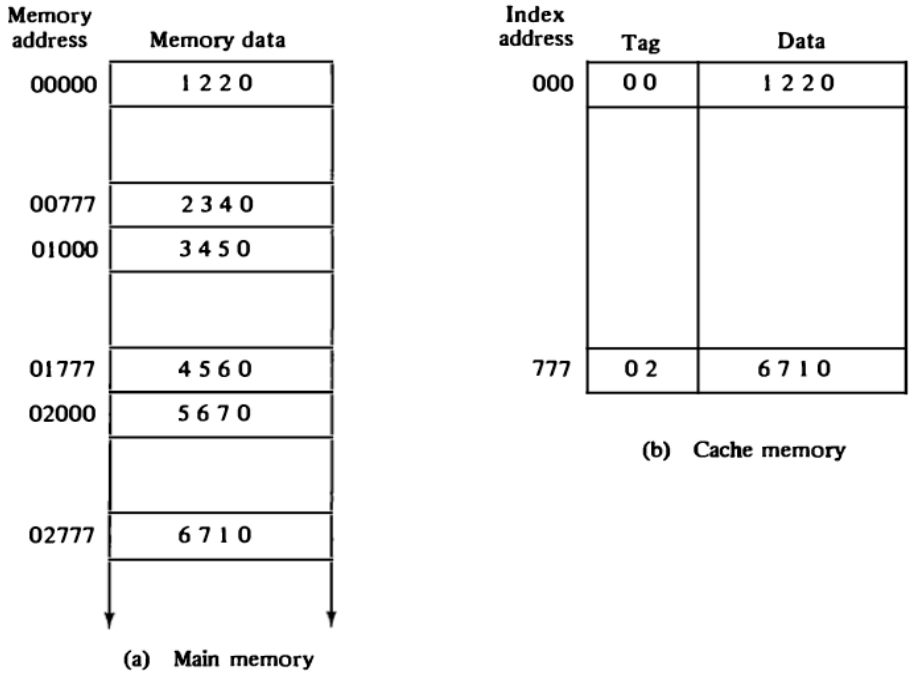


Figure 13: Direct mapping cache organization

- Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored along the data bits.
- When the CPU generates a memory request, the index field is used for the address to access the cache.
- The *tag* field of the CPU address is compared with the tag in the word read from the cache.

→If the two tags match, there is a *hit* and the desired data word is in cache.

- Eg: Suppose that the CPU now wants to access the word at address 02777. So the index value is 777 and tag value is 02.
- The index value 777 is used as address to access the cache memory. In the above figure 13, index address 777 is available, and then tag value 02 is compared with tag value in the cache memory associated with index address 777.
- Here tag value of CPU address is match with tag value of cache with index address also. So desired data is available in cache memory. So it is hit operation.

→ If there is no match, there is a *miss* and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value.

- Eg: Suppose that the CPU now wants to access the word at address 01777. So the index value is 777 and tag value is 01.
- The index value 777 is used as address to access the cache memory. In the above figure 13, index address 777 is available, and then tag value 01 is compared with tag value in the cache memory associated with index address 777.
- Here tag value of CPU address 01 is not match with tag value of cache memory is 02. So desired data was not available in cache memory. So it is *miss* operation.
- Whenever a miss operation was occur, the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value.

- The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags.

3.5.3 Set-Associative Mapping

- To overcome the disadvantage of direct mapping, i.e. two words with the same index in their address but with different tag values cannot reside in cache memory at the same time, Set-associative mapping is proposed.
- Here each word of cache can store two or more words stored together with its tag.
- An example of a set-associative cache organization for a set size of two is shown in Figure 14.

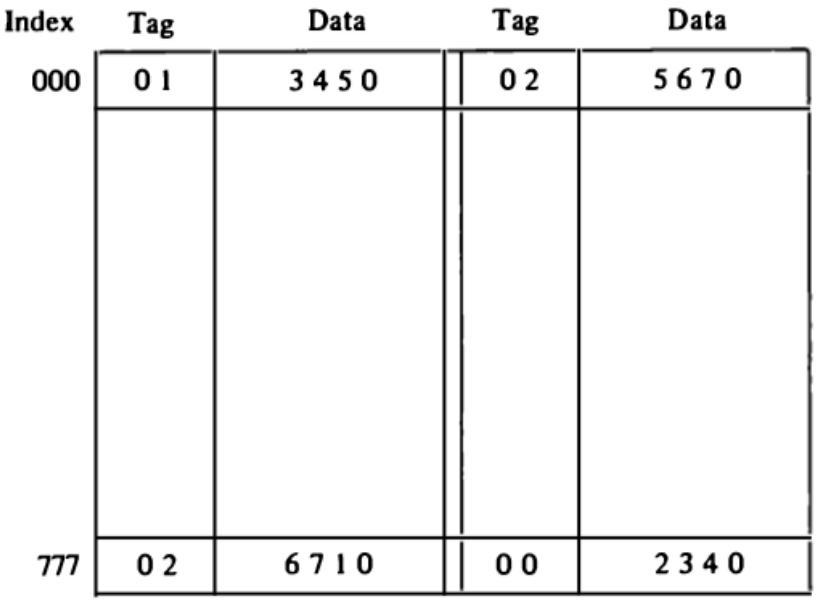


Figure 14: Two way set-associative mapping cache

- In the above figure 14, each index address refers to two data words and their associated tags.
- Here each tag requires 6 bits, i.e. 2 octal digits and 12 bits for data, i.e. 4 octal digits.
- In the above figure 14, the words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000 with different tag values 01 and 02 respectively.
- Similarly, the words at addresses 02777 and 00777 of main memory are stored in cache memory at index address 777 with different tag values 01 and 02 respectively.
- When the CPU generates a memory request, the index value of the address is used to access the cache.
- The tag field of the CPU address is then compared with both tags in the cache. If tags are match then it is a *hit* operation.
- If the tag field of the CPU address and tags in the cache are not matched then it is a *miss* operation.
- When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value.

3.5.4 Writing into Cache

- An important aspect of cache organization is concerned with memory write requests.
- There are two ways for writing into Cache.
- The simplest and most commonly used procedure is to update main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the

specified address. This is called the *write-through* method. This method has the advantage that main memory always contains the same data as the cache.

- The second procedure is called the write-back method. In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is updated into main memory.

3.5.5 Cache Initialization

- The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory.
- After initialization the cache is considered to be empty, but in effect it contains some non-valid data. It is customary to include with each word in cache a *valid bit* to indicate whether or not the word contains valid data.
- The cache is initialized by clearing all the valid bits to 0.

3.6 VIRTUAL MEMORY

- In memory hierarchy system, programs and data first stored in Auxiliary memory.
- Position of data or programs are brought into main memory as they needed by the CPU.
- Virtual memory is a concept used in some large computers that permit the users to construct programs as though a large memory space were available, equal to the totality of Auxiliary memory.
- Virtual memory is used to give programmers as the illusion that they have large memory even though computer actually has a relatively small main memory.
- A virtual memory system provides a mechanism for translating program generated address into correct main memory location.
- Address space: An address used by a programmer will be called a virtual address, and the set of virtual address called as address space.
- Memory space: An address in main memory (physical memory) is called a location (or) physical address, and set of physical addresses is called as memory space.
- The address space allowed is larger than the memory space in computer with virtual memory.

Unit -III

Assignment-Cum-Tutorial Questions

Section - A

1. DRAM is used in implementing the_____ [ ]
   a. Secondary memory c. Static memory
   b. Dynamic memory d. Main memory
2. ROM is a type of _____memory that is capable of holding data. [ ]
   a. Random Access c. add on
   b. Plug and play d. Built in
3. Which type of memory is used for increasing the speed? [ ]
   a. Memory hierarchy c. Virtual memory
   b. Cache memory d. Memory system
4. Performance of cache memory is measured in terms of quantity called as _____ [ ]
   a. Hit ratio c. Miss ratio
   b. Count ratio d. Bit ratio
5. Cache memory works on the principle of_____ [ ]
   a. Locality of data c. Locality of memory
   b. Locality of reference d. Locality of reference & memory
6. Ratio of cache accesses, results in a miss is known as_____ [ ]
   a. Hit miss c. Hit rate
   b. File caches d. Miss rate
7. During a write operation if the required block is not present in the cache then _____ occurs.
   a. Write latency c. Write hit [ ]
   b. Write delay d. Write miss
8. Cache memory acts between_____ [ ]
   a. CPU and RAM c. RAM and ROM
   b. CPU and Hard Disk d. None of these
9. Write Through technique is used in which memory for updating the data _____ [ ]
   a. Virtual memory c. Main memory
   b. Auxiliary memory d. Cache memory
10. The cache memory of 2k words uses direct mapping with a block size of 4 words. Find out the number of blocks cache can accommodate? [ ]
    a. 512 words c. 256 words

c. 128 words d. 1024 words

11. If the main memory is of 8K bytes and the cache memory is of 2K words. It uses associative mapping. Then each word of cache memory shall be_____. [ ]
    a. 11 bits c. 21 bits
    b. 16 bits d. 20 bits
12. While using the direct mapping technique, in a 16 bit system the higher order 5 bits is used for _____. [ ]
    a. Tag c. Block
    b. Word d. Id
13. Which memory is used to hold programs for future use_____ [ ]
    a. Auxiliary memory c. Main memory
    b. Cache memory d. Virtual memory
14. Which type of disks is permanently attached to the unit assembly and cannot be removed by the occasional user? [ ]
    a. Hard disk c. Floppy
    c. DVD d. Memory card
15. In virtual memory concept an address used by a programmer is called as _____ [ ]
    a. virtual address c. Physical address
    b. blocks d. Memory space
16. In the memory hierarchy, as the speed of operation increases the memory size also increases.
    a. True b. False [ ]

Section -B

1. Derive the expression for Match Logic and implement it using logic gates.
2. Describe how Read and Write operations are performed in Associative Memory.
3. With a suitable diagram explain associative memory.
4. Explain different types of mapping functions in cache memory.
5. Compare and contrast direct and associative mapping techniques.
6. Differentiate Cache Memory Vs Virtual Memory.
7. Determine the efficiency of hierarchical organization of memory in terms of speed, size and Cost.
8. A computer system has a cache with T_c=10 ns and a hit ratio is 80 percent and the average memory access time TM=240 ns .What is the access time for Physical memory?
9. A computer system has a cache with T_c=8 ns and Physical memory with T_p=65 ns. If the hit ratio is 75 percent, what is the average memory access time?

10. A computer system has a cache with T_c=10 ns and Physical memory with T_p=55 ns and physical memory with access time TM=40 ns. What is the hit ratio?

Section – C

1. A computer has a 256 KByte, 4-way set associative, write back data cache with block size of 32 Bytes. The processor sends 32 bit addresses to the cache controller. Each cache tag directory entry contains, in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit. The number of bits in the tag field of an address is____ **(GATE CS 2012)** [ ]

   a. 11 b. 14 c. 16 d.27

2. Consider the data given in previous question. The size of the cache tag directory is_____ **(GATE CS 2012)** [ ]

   a. 160 Kbits b. 136 bits c. 40 Kbits d. 32 bits

3. An 8KB direct-mapped write-back cache is organized as multiple blocks, each of size 32-bytes. The processor generates 32-bit addresses. The cache controller maintains the tag information for each cache block comprising of the following. 1 Valid bit and 1 Modified bit. As many bits as the minimum needed to identify the memory block mapped in the cache. What is the total size of memory needed at the cache controller to store meta-data (tags) for the cache?**(GATE CS 2011)**[ ]

   a. 4864 bits b. 6144 bits c. 6656 bits d. 5376 bits

4. Consider a 4-way set associative cache consisting of 128 lines with a line size of 64 words. The CPU generates a 20-bit address of a word in main memory. The number of bits in the TAG, LINE and WORD fields are respectively? **(GATE-CS-2007)** [ ]

   a. 9, 6, 5 b. 7, 7, 6 c. 7, 5, 8 d.9, 5, 6

5. Consider two cache organizations: The first one is 32 KB 2-way set associative with 32-byte block size. The second one is of the same size but direct mapped. The size of an address is 32 bits in both cases. A 2-to-1 multiplexer has a latency of 0.6 ns while a kbit comparator has a latency of k/10 ns. The hit latency of the set associative organization is h1 while that of the direct mapped one is h2. The value of h1 is? **(GATE-CS-2006)** [ ]

   a. 2.4 ns b. 2.3 ns c. 1.8 ns d.1.7 ns

6. Consider two cache organizations: The first one is 32 KB 2-way set associative with 32-byte block size. The second one is of the same size but direct mapped. The size of an address is 32 bits in both cases. A 2-to-1 multiplexer has a latency of 0.6 ns while a kbit comparator has a latency of k/10 ns. The hit latency of the set associative organization is h1 while that of the direct mapped one is h2. The value of h2 is? **(GATE-CS-2006)** [ ]

a. 2.4 ns b. 2.3 c. 1.8 d. 1.7

7. Consider a direct mapped cache of size 32 KB with block size 32 bytes. The CPU generates 32 bit addresses. The number of bits needed for cache indexing and the number of tag bits are respectively? **(GATE-CS-2005)** [ ]

a. 10, 17 b. 10, 22 c. 15, 17 d. 5, 17

8. A cache line is 64 bytes. The main memory has latency 32ns and bandwidth 1G.Bytes/s. The time required to fetch the entire cache line from the main memory is? **(GATE IT 2006)** [ ]

a. 32 ns b. 64 ns c. 96 ns d. 128 ns

UNIT – IV

COMPUTER ARITHMETIC

Objective:

- To familiarize with the algorithms to perform arithmetic operations.

Syllabus:

Data representation- fixed point, floating point, addition and subtraction, multiplication and division algorithms.

Learning Outcomes:

At the end of the unit student will be able to:

- Represent data in fixed and floating point formats.
- Perform arithmetic operations on data represented in fixed point, floating point formats.

Learning Material

4.1 DATA REPRESENTATION

- For positive numbers sign bit equal to 0 and 1 for negative numbers.
- In addition to the sign, a number may have a binary (or decimal) point.
- The position of the binary point is needed to represent fractions, integers, or mixed integer-fraction numbers.
- There are two ways of specifying the position of the binary point in a register: by giving it a fixed position or by employing a floating-point representation.
- The fixed-point method assumes that the binary point is always fixed in one position. The two positions most widely used are:
    1. A binary point in the extreme left of the register to make the stored number a fraction.
    2. A binary point in the extreme right of the register to make the stored number an integer.
- In either case, the binary point is not actually present, but its presence is assumed from the fact that the number stored in the register is treated as a fraction or as an integer.

4.1.1 Integer Representation

- When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number.
- When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

1. Signed Magnitude representation
2. Signed 1's complement representation
3. Signed 2's complement representation

Eg: There is only one way to represent + 14, there are three different ways to represent -14 with eight bits.

In signed-magnitude representation	0 0001110	+14
In signed-magnitude representation	1 0001110	-14
In signed-1's complement representation	1 1110001	-14
In signed-2's complement representation	1 1110010	-14

4.1.2 Arithmetic Addition

Signed-magnitude Addition

- The addition of two numbers in the ***signed-magnitude*** system follows the rules of ordinary arithmetic.
- If the signs are the same, we add the two magnitudes and give the sum the common sign.
- If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.
- For example, (+25) + (-37) = -(37 - 25) = -12 and is done by subtracting the smaller magnitude 25 from the larger magnitude 37 and using the sign of 37 for the sign of the result.

Signed 2's complement addition

- The addition of two numbers in the signed 2's complement addition system follows.
- Add the two numbers, including their sign bits, and discard any carry out of the sign (leftmost) bit position. Numerical examples for addition are shown below.

Eg 1:

+6	00000110
+13	00001101
+19	00010011

Eg 2:

-6	11111010	(Signed 2's complement of -6)
+13	00001101	
+7	1]00000111	

Eg3:

+6	00000110	
-13	11110011	(Signed 2's complement of -13)
-7	11111001	(Signed 2's complement of -7)

Eg4:

-6	11111010	(Signed 2's complement of -6)
-13	11110011	(Signed 2's complement of -13)
-19	1] 11101101	(Signed 2's complement of -19)

4.1.3 Arithmetic Subtraction

- Subtraction of two signed binary numbers when negative numbers are in 2's complement form is very simple and can be stated as follows.
- Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit). A carry out of the sign bit position is discarded.

Eg 1:

-6	11111010	(Signed 2's complement of -6)
-13	11110011	(Signed 2's complement of -13)

-6	11111010	
-13	00001101	(2's complement of -13)
+7	1] 00000111	

Eg 2:

+13	00001101	
+6	00000110	

+13	00001101	
+6	11111010	(2's complement of +6)
+7	1] 00000111	

Eg 3:

-6	11111010	(Signed 2's complement of -6)
+13	00001101	

-6	11111010	
+13	11110011	(2's complement of +13)
-19	1] 11101101	

Eg 4:

-13	11110011	(Signed 2's complement of -13)
-6	11111010	(Signed 2's complement of -6)

-13	11110011	
-6	00000110	(2's complement of -6)
-7	11111001	(Signed 2's complement of -7)

** When ever Addition or Subtraction operation is performed on two numbers which are represented using signed 2's complement form, if result is –ve number, i.e. MSB is 1, then that indicates result is also in Signed 2's complement form.

4.1.4 Overflow

- When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an overflow occurred.
- An overflow is a problem in digital computers because the width of registers is finite and fixed. A result that contains n + 1 bits cannot be accommodated in a register with a standard length of n bits. For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set which can then be checked by the user.
- An overflow cannot occur after an addition if one number is positive and the other is negative.

carries: 0 1		carries: 1 0	
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010

- An overflow condition can be detected by observing the carry into the sign bit position and the carry out of the sign bit position.
- If these two carries are *not equal*, an overflow *condition is produced*.
- If these two carries are *equal*, an overflow *condition is not produced*.

4.2 ADDITION AND SUBTRACTION

- There are three ways of representing negative fixed-point binary numbers:
  1. signed-magnitude
  2. Signed-l's complement
  3. Signed-2's complement

4.2.1 Addition and Subtraction with Signed-Magnitude Data

- The algorithms for addition and subtraction stated as follows (the words inside parentheses should be used for the *subtraction* algorithm):
- Addition (subtraction) algorithm.

  1. when the signs of A and B are identical (*different*), add the two magnitudes and attach the sign of A to the result.

  2. 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.

Table 1: Addition and Subtraction of Signed Magnitude Numbers

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

4.2.2 Hardware Implementation for Addition and Subtraction with Signed-Magnitude Data

- To implement the two arithmetic operations with hardware, it is first necessary that the two numbers be stored in registers.
- Let A and B be two registers that hold the magnitudes of the numbers, and A_s, and B_s be two flip-flops that hold the corresponding signs.
- Here parallel-adder is needed to perform the microoperation A + B. (Consists of Full adder).
- The complementer for generating the 2's Complement while performing subtraction operation. (Consists of X-OR gate).

- Where M is Mode of operation. When M = 0, the output of B is transferred to the adder, the input carry is 0, and the adder is equal to the sum A + B.
- When M =1, the l's complement of B is applied to the adder, the input carry is 1, and output is = $A + \bar{B} + 1$. This is equal to A plus the 2's complement of B, which is equivalent to the A – B.

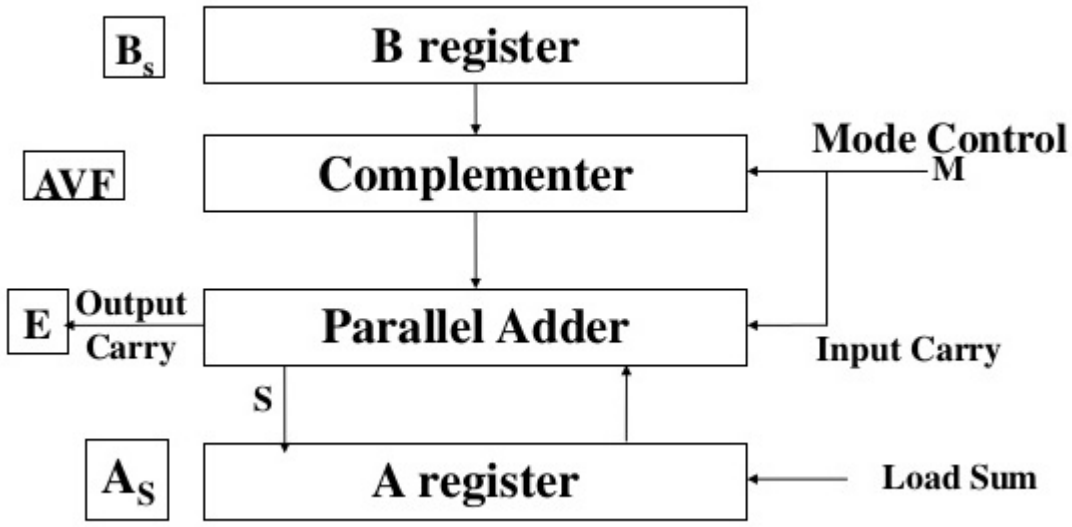


Figure 1: Hardware for signed-magnitude addition and subtraction

4.2.3 Hardware Algorithm for Signed-magnitude addition and subtraction

- The magnitudes are added with a microoperation EA←A + B. where EA is a register that combines E and A.
- The value of E is transferred into the Add-overflow flip-flop AVF, if E is 1.
- The magnitudes are subtracted by adding A to the 2's complement of B.
- No overflow can occur if the numbers are subtracted so AVF is cleared to 0.
- 1 in E indicates that A≥ B and the number in A is the correct result. If this number i.s zero, the sign A, must be made positive to avoid a negative zero.
- 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. This operation can be done with one mlcrooperation $A \leftarrow \bar{A} + 1$.

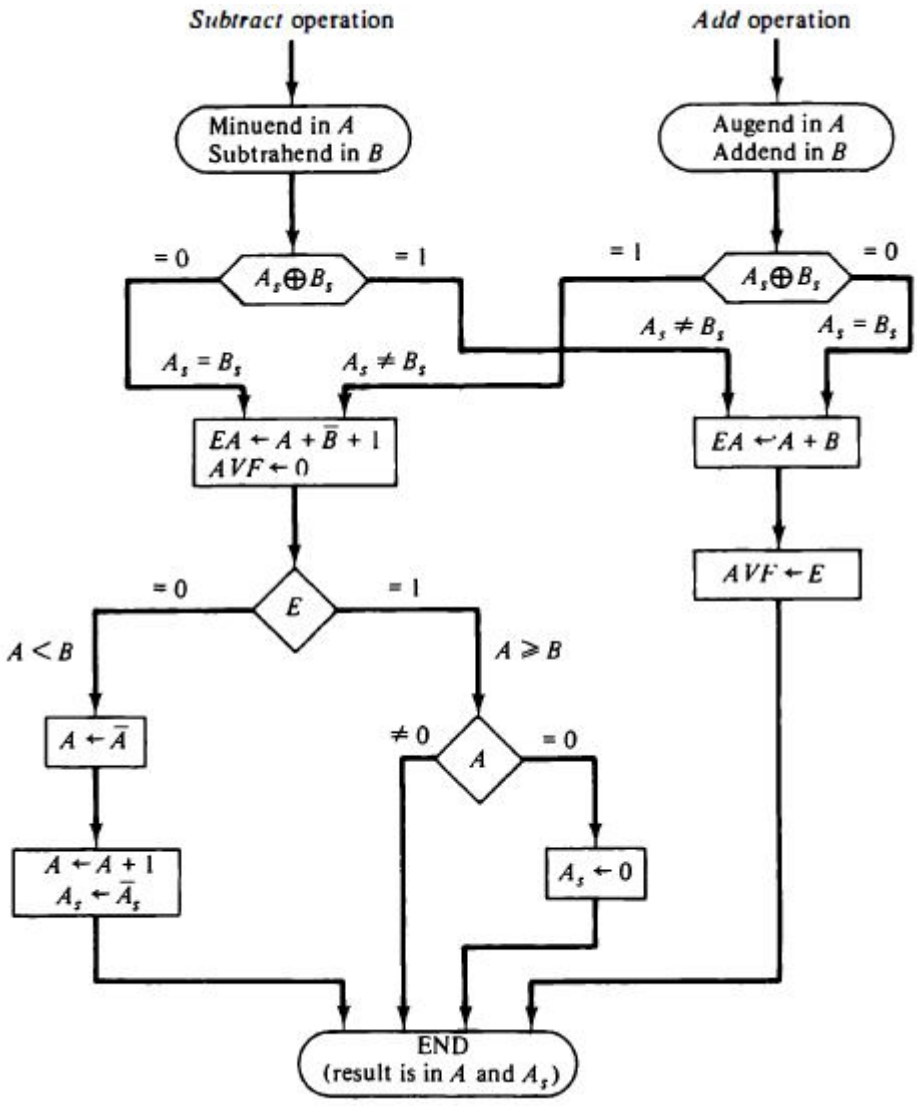


Figure 2: Flowchart for add and subtract operations

4.2.4 Addition and Subtraction with Signed-2's Complement Data

- 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.
- The register configuration for the hardware implementation is shown in Figure 3.

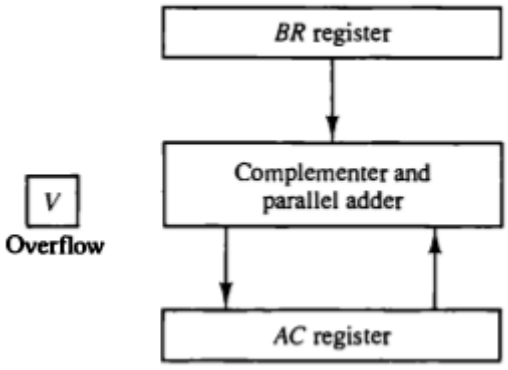


Figure 3: Hardware for signed2's complement addition and subtraction

- The algorithm for adding and subtracting two binary numbers in signed2's complement representation is shown in the flowchart of Figure 4.

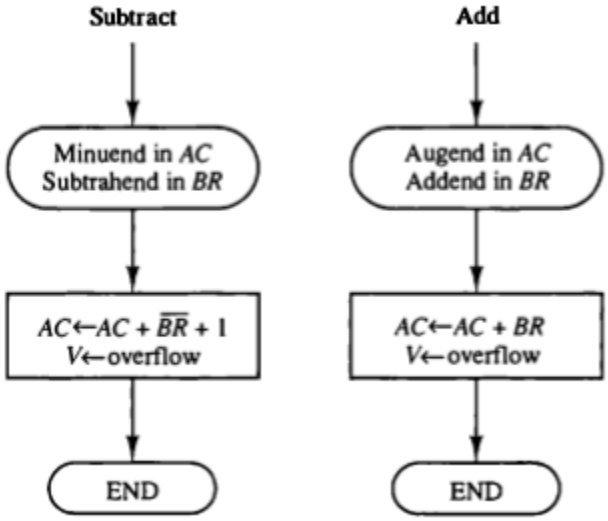


Figure 4: Algorithm for adding and subtracting numbers in signed 2's complement representation

- 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's complement of BR. Taking the 2's complement of BR has the effect of changing a positive number to negative, and vice versa. An overflow must be checked during this operation because the two numbers added could have the same sign.
- The programmer must realize that if an overflow occurs, there will be an erroneous result in the AC register.

4.3 MULTIPLICATION ALGORITHMS

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

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

4.3.1 Hardware Implementation for Signed-Magnitude Data Multiplication

- The hardware for multiplication consists of the equipment shown in Figure 5.

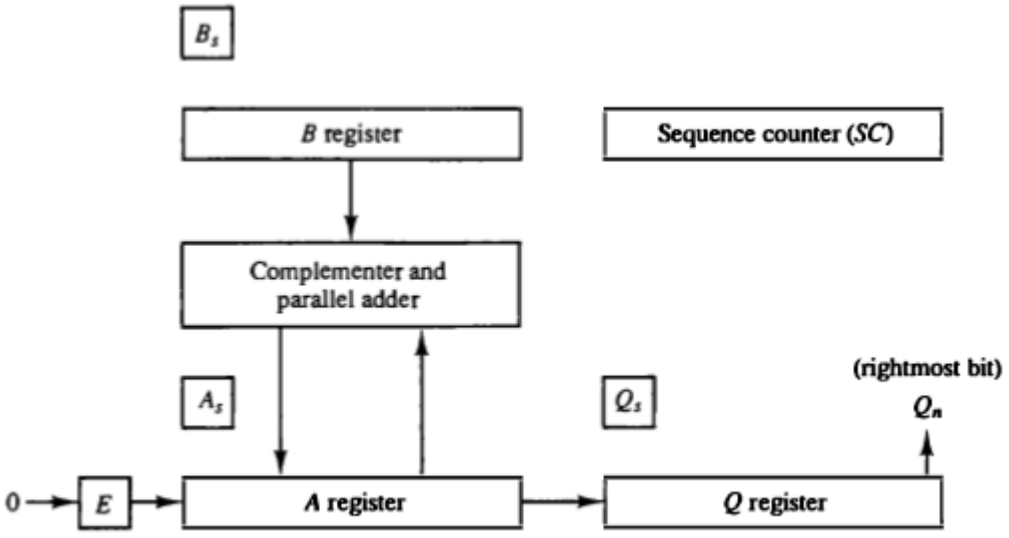


Figure 5: Hardware for multiply operation

- Initially, the multiplicand is in register B and the multiplier in Q.
- Initially A is set to 0 as number of bits in the multiplicand.
- The sequence counter SC is initially set to a number equal to the number of bits in the multiplier.
- 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 5.

- 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_n, will hold the bit of the multiplier, which must be inspected next.
- 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.

4.3.2 Hardware Algorithm for Signed-Magnitude Data Multiplication

- The following figure 6 is a flowchart of the hardware multiply algorithm.

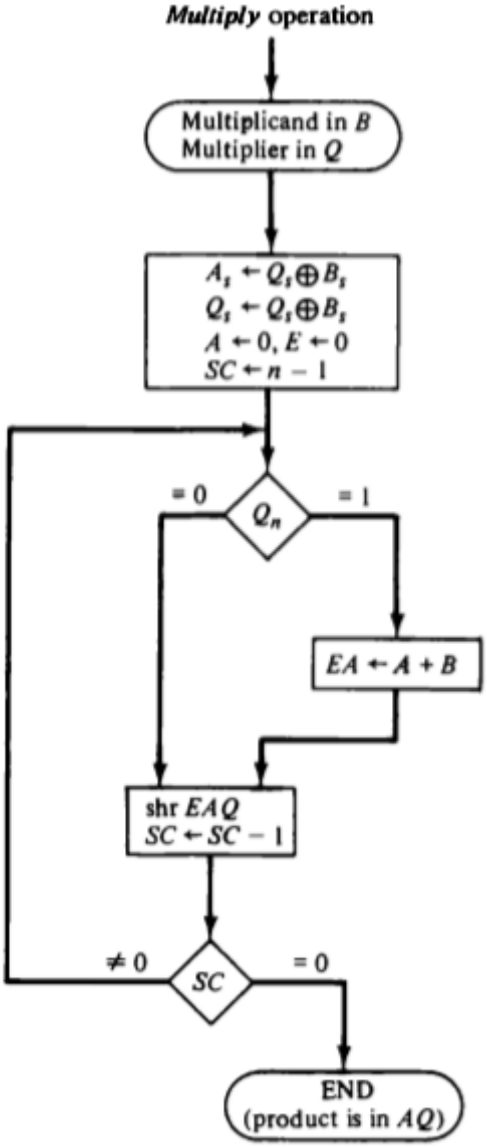


Figure 6: Flowchart multiply operation on sign magnitude representation numbers

- Initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in B_s and Q_s, respectively.
- The signs are compared, and both signs of 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.
- After the initialization, the low-order bit of the multiplier in Q, is tested. If it is a 1, the multiplicand in B is added to the present partial product in A. If it is a 0, nothing is done.
- Register EAQ is then shifted once to the right to form the new partial product. The sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated and a new partial product is formed. The process stops when SC = 0.
- Note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces the multiplier. The final product is available in both A and Q, with A holding the most significant bits and Q holding the least significant bits.
- The following table describes multiplication of binary numbers 10111(+23) and 10011(+19) which are represented using Sign Magnitude Representation.

Table 2: Numerical Example for Binary Multiplier

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				

Now Result is available in Registers A and Q. i.e. 0110110101 => 437 and sign bit of A is 0. So result is +437.

- The following table 3 describes multiplication of binary numbers **10011(+19)** and **00110(+6)** which are represented using Sign Magnitude Representation.

- Here Multiplicand is positive value, so $B_s = 0$. Here Multiplier is positive value, so $Q_s = 0$.
- Now $A_s = B_s \oplus Q_s$, i.e $A_s = 0 \oplus 0 \Rightarrow 0$.

Table 3: Numerical Example for Binary Multiplier

Multiplicand B=10011	**E**	**A**	**Q**	**SC**
Initially	0	00000	00110	5
$Q_n = 0$. So shr EAQ	0	00000	00011	4
$Q_n = 1$. So add B to A		00000 10011 10011		
Now shr EAQ	0	01001	10001	3
$Q_n = 1$. So add B to A		01001 10011 11100		
Now shr EAQ	0	01110	01000	2
$Q_n = 0$. So shr EAQ	0	00111	00100	1
$Q_n = 0$. So shr EAQ	0	**00011**	**10010**	0

Now Result is available in Registers A and Q. i.e. 0001110010 => 114 and sign bit of A is 0. So result is +114.

- The following table 4 describes multiplication of binary numbers **10010(-18)** and **00110(+5)** which are represented using Sign Magnitude Representation.
- Here Multiplicand is negative value, so $B_s = 1$. Here Multiplier is positive value, so $Q_s = 0$.
- Now $A_s = B_s \oplus Q_s$,i.e $A_s = 1 \oplus 0 \Rightarrow 1$.

Table 4: Numerical Example for Binary Multiplier

Multiplicand B=10010	E	A	Q	SC
Initially	0	00000	00101	5
Q_n = 1. So add B to A		00000 10010 10010		
Now shr EAQ	0	01001	00010	4
Q_n = 0. So shr EAQ	0	00100	10001	3
Q_n = 1. So add B to A		00100 10010 10110		
Now shr EAQ	0	01011	01000	2
Q_n = 0. So shr EAQ	0	00101	10100	1
Q_n = 0. So shr EAQ	0	**00010**	**11010**	0

Now Result is available in Registers A and Q. i.e. 0001011010 => 90 and sign bit of A is 1.
So result is **-90**.

4.3.3 Booth Multiplication Algorithm (for signed-2's complement numbers)

- Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.
- As in all multiplication schemes, Booth algorithm requires examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to the following rules:
  1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
  2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of 0's in the multiplier.
  3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.
- The hardware implementation of Booth algorithm requires the register configuration shown in Figure 7.

- Q_n designates the least significant bit of the multiplier in register QR. An extra flip-flop $Q_{n+1}$is appended to QR to facilitate a double bit inspection of the multiplier.

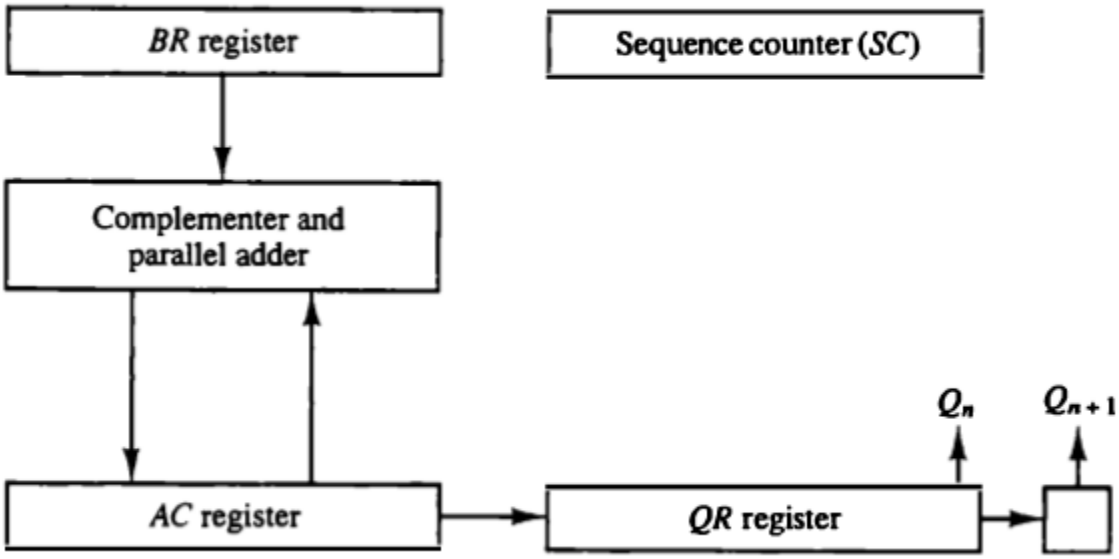


Figure 7: Hardware for Booth algorithm

- The flowchart for Booth algorithm is shown in Figure 8.
- AC and the appended bit Q_{n+1} are initially cleared to 0 and the sequence counter SC is set to a number n equal to the number of bits in the multiplier.
- The two bits of the multiplier in Q_n and Q_{n+1} are inspected.
- If the two bits are equal to 10, it means that the first 1 in a string of 1's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.
- If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.
- When the two bits are equal, the partial product does not change. An *overflow cannot* occur because the addition and subtraction of the multiplicand follow each other.
- The next step is to shift right the partial product and the multiplier (including bit Q_{n+1}). This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC unchanged.
- The sequence counter is decremented and the computational loop is repeated n times.

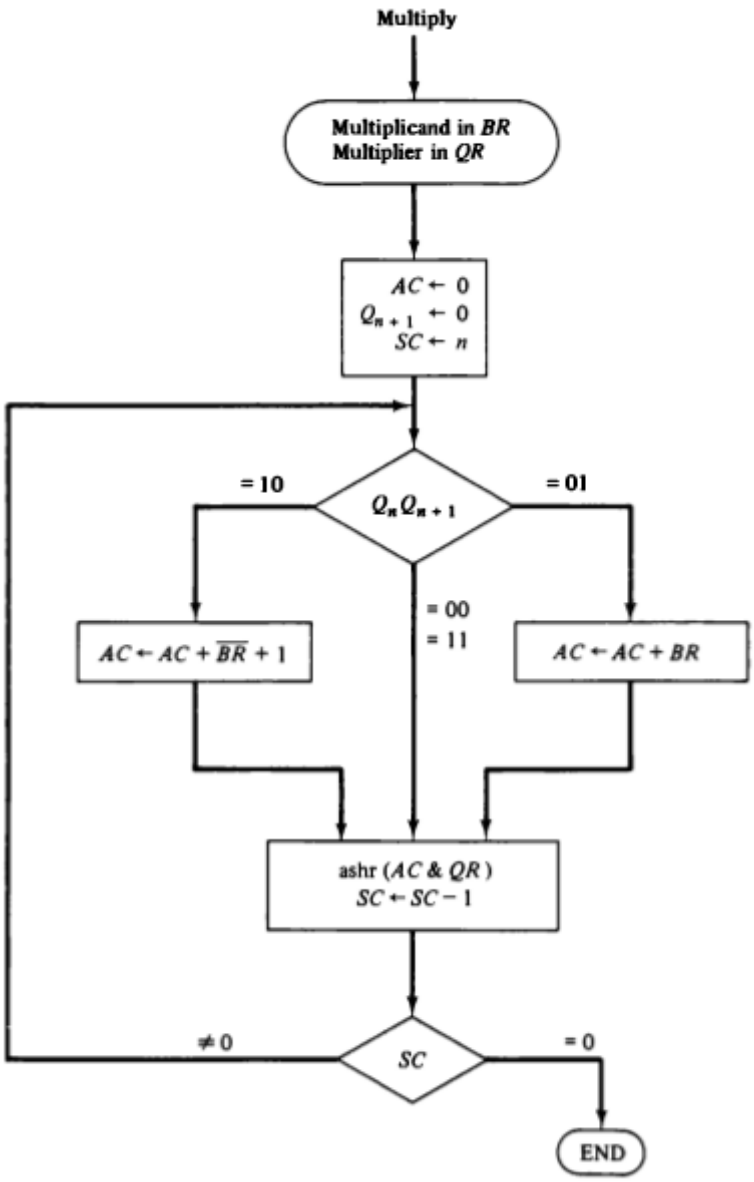


Figure 8: Booth algorithm for multiplication of signed 2's complement numbers.

- A numerical example of Booth algorithm is shown in Table 5. It shows the step-by-step multiplication of (-9) x (-13) = + 117.
- Here the multiplier in QR is negative and that the multiplicand in BR is also negative. The 10-bit product appears in AC and QR and is positive.

Table 5: Example of Multiplication with Booth Algorithm

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	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

Now Result is available in Registers AR and QR. i.e. 0001110101 => +117.

4.4 DIVISION ALGORITHMS

- The division process is illustrated by a numerical example in Figure 9. The divisor B consists of five bits and the dividend A, of ten bits.

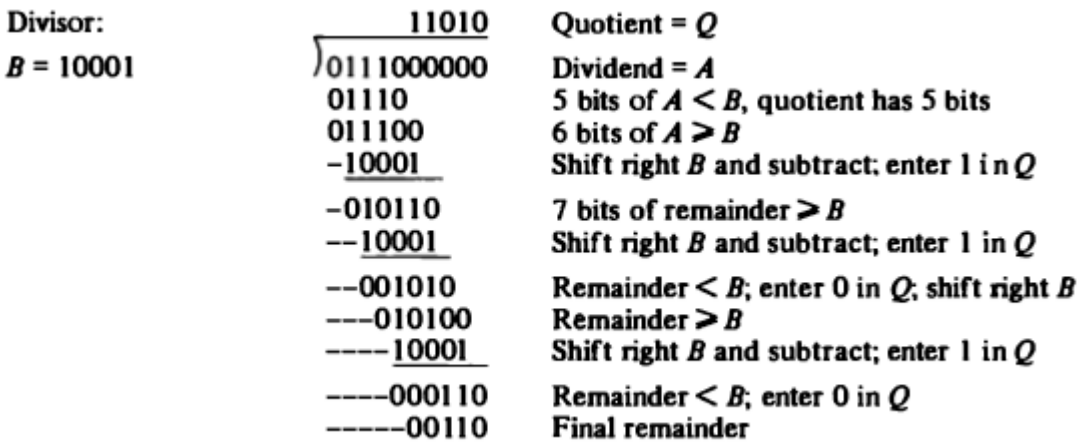


Figure 9: Example of binary division

- The five most significant bits of the dividend are compared with the divisor. Since the 5-bit number is smaller than B, we try again by taking the six most significant bits of A and compare this number with B.

- The 6-bit number is greater than B, so we place a 1 for the quotient bit in the sixth position above the dividend. The divisor is then shifted once to the right and subtracted from the dividend. The difference is called a partial remainder.
- The process is continued by comparing a partial remainder with the divisor. If the partial remainder is greater than or equal to the divisor, the quotient bit equal to 1. The divisor is then shifted right and subtracted from the partial remainder.
- If the partial remainder is smaller than the divisor, the quotient bit is 0 and no subtraction is needed. The divisor is shifted once to the right in any case. Note that the result gives both a quotient and a remainder.

4.4.1 Hardware Implementation for Signed-Magnitude Data Division operation

- When the division is implemented in a digital computer, it is convenient to change the process slightly. Instead of shifting the divisor to the right, the dividend, or partial remainder, is shifted to the left.
- Subtraction may be achieved by adding A to the 2's complement of B.
- The hardware for implementing the division operation is identical to that required for multiplication. Register EAQ is now *shifted to the left* with 0 inserted into Q, and the previous value of E lost.

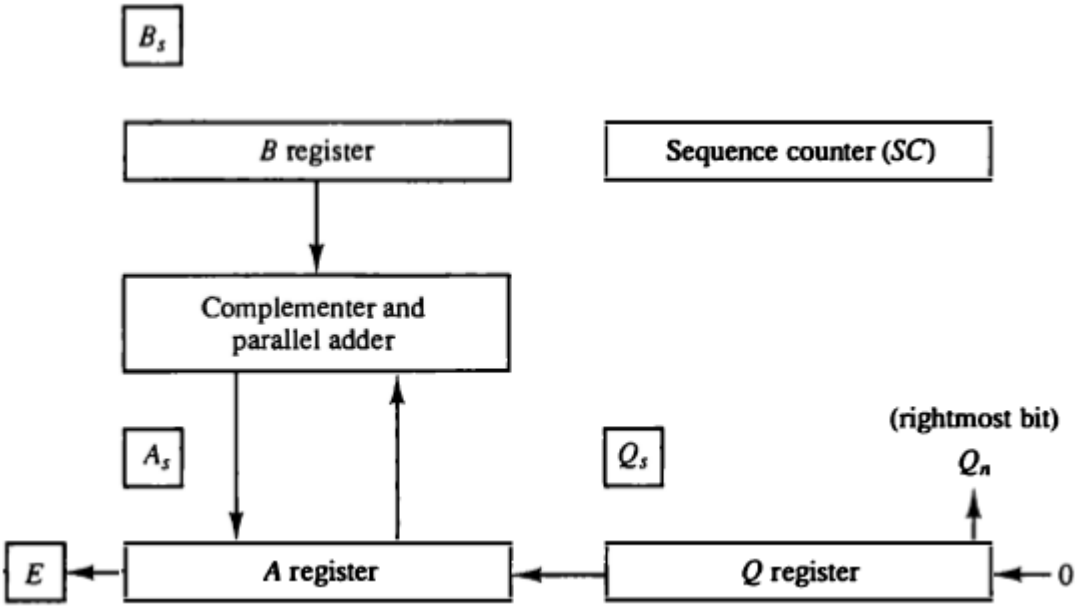


Figure 10: Hardware for division operation

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

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

Figure 11: Example of binary division with digital hardware

- The divisor is stored in register B and the double-length dividend is stored in registers A and Q.
- The dividend is shifted to the left and the divisor is subtracted by adding its 2's complement value. The information about the relative magnitude is available in E.
- If E = 1, it signifies that A ≥ B. 1 is inserted into Q_n(LSB of Quotient), 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 Q_n, remains 0 (inserted during the shift). The value of B is then added to restore the partial remainder in A to its previous value.
- The partial remainder is shifted to the left and the process is repeated again until the partial remainder obtained is smaller than the divisor.
- Finally, the quotient is in Q and the final remainder is in A.

- The sign of the quotient is determined from the signs of the dividend and the divisor. If the two signs are same, the sign of the quotient is plus. If they are different, the sign is minus. The sign of the remainder is the same as the sign of the dividend.

4.4.2 Divide Overflow

- The division operation may result in a quotient with an overflow.
- When the dividend is twice as long as the divisor, the condition for overflow can be stated as follows: A divide-overflow condition occurs if the high-order half bits of the dividend constitute a number greater than or equal to the divisor.
- Another problem associated with division is the fact that a division by zero must be avoided. This occurs because any dividend will be greater than or equal to a divisor which is equal to zero.
- Overflow condition is usually detected when a special flip-flop is set. We will call it a divide-overflow flip-flop and label it DVF.

4.4.3 Hardware Algorithm

- The hardware divide algorithm is shown in the flowchart of Figure 12. The dividend is in A and Q and the divisor in B. The sign of the result is transferred into Q_s, to be part of the quotient.
- A constant is set into the sequence counter SC to specify the number of bits in the quotient.
- A divide-overflow condition is tested by subtracting the divisor in B from half of the bits of the dividend stored in A.
- If A≥B, the divide-overflow flip-flop DVF is set and the operation is terminated prematurely. If A < B, no divide overflow occurs so the value of the dividend is restored by adding B to A.
- The division of the magnitudes starts by shifting the dividend in AQ to the left with the high-order bit shifted into E. If the bit shifted into E is 1, we know that EA > B because EA consists of a 1 followed by n-1 bits while B consists of only n-1 bits. In this case, B must be subtracted from EA and 1 inserted into Q_n, for the quotient bit.
- If the shift-left operation inserts a 0 into E, the divisor is subtracted by adding its 2's complement value and the carry is transferred into E.
- If E = 1, it signifies that A ≥ B; therefore, Q, is set to 1.
- If E = 0, it signifies that A < B and the original number is restored by adding B to A.
- This process is repeated again with register A holding the partial remainder. After n - 1 times, the quotient magnitude is formed in register Q and the remainder is found in register A.
- The quotient sign is in Q_s, and the sign of the remainder in A_s, is the same as the original sign of the dividend.

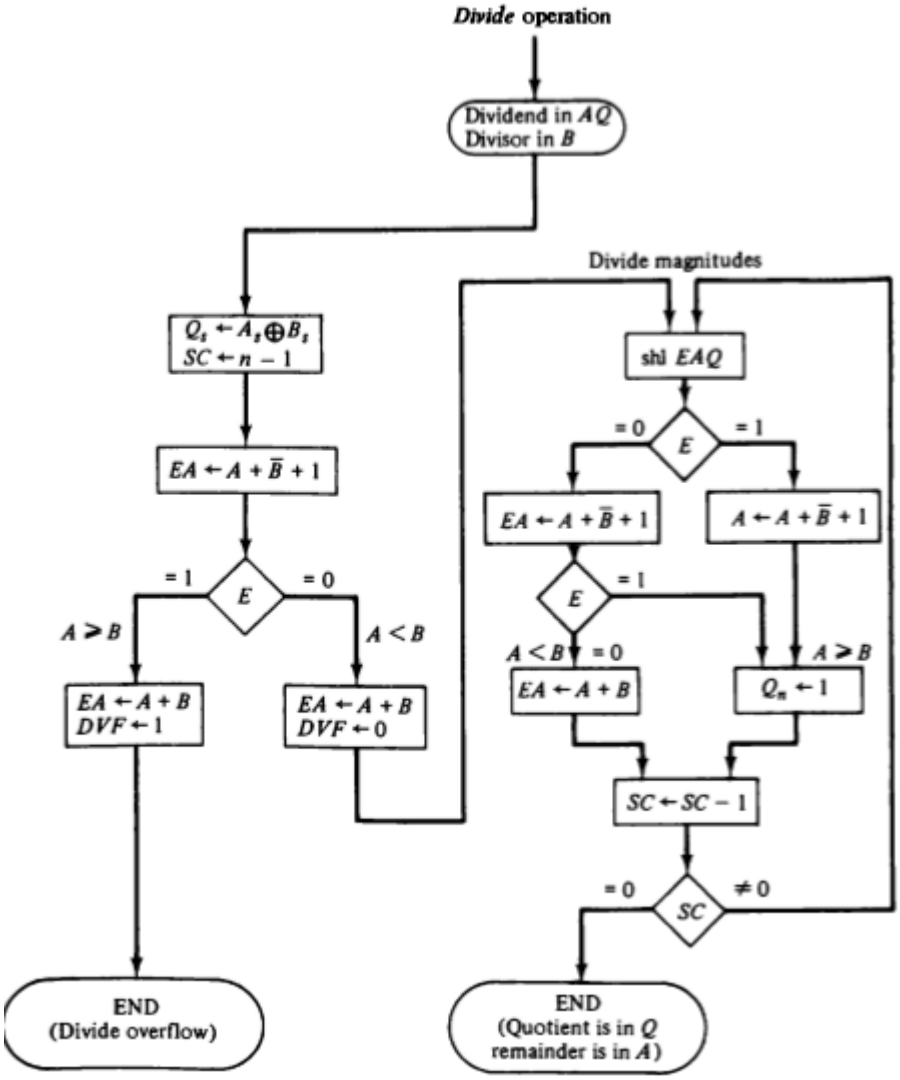


Figure 12: Flowchart for divide operation

4.4.4 Other Algorithms

- The hardware method just described is called the *restoring method*. The reason for this name is that the partial remainder is restored by adding the divisor to the negative difference.
- Two other methods are available for dividing numbers, the *comparison method* and the *non-restoring method*.
- In the *comparison method* A and B are compared prior to the subtraction operation. Then if $A \geq B$, B is subtracted from A. If $A < B$ nothing is done.
- The partial remainder is shifted left and the numbers are compared again. The comparison can be determined prior to the subtraction by inspecting the end-carry out of the parallel-adder prior to its transfer to register E.

- In the *non-restoring method*, B is not added if the difference is negative but instead, the negative difference is shifted left and then B is added. To see why this is possible consider the case when A < B. From the flowchart in Figure 12, we note that the operations performed are A - B + B; that is, B is subtracted and then added to restore A. The next time around the loop, this number is shifted left (or multiplied by 2) and B subtracted again. This gives 2(A - B + B) - B = 2A - B.
- This result is obtained in the non-restoring method by leaving A – B as it is. The next time around the loop, the number is shifted left and B added to give 2(A - B) + B = 2A - B, which is the same as before.
- Thus, in the non-restoring method, B is subtracted if the previous value of Q_n was a 1, but B is added if the previous value of Q_n was a 0 and no restoring of the partial remainder is required.
- This process saves the step of adding the divisor if A is less than B, but it requires special control logic to remember the previous result. The first time the dividend is shifted, B must be subtracted. Also, if the last bit of the quotient is 0, the partial remainder must be restored to obtain the correct final remainder.

4.5 FLOATING-POINT REPRESENTATION

- A floating point number in computer registers consists of two parts: a mantissa m and an exponent e.
- The two parts represent a number obtained from multiplying m times a radix r raised to the value of e.

$$\mathbf{m \times r^e}$$

- The mantissa may be a fraction or an integer. The location of the radix point and the value of the radix r are assumed and are not included in the registers.
- For eg, the decimal number +6132.789 is represented in floating point with a fraction and exponent as follows:

Fraction	Exponent
+0.6132789	+04

- The above representation is equivalent to scientific notation +0.6132789 X 10^4
- A floating point binary number is represented in a similar manner to floating point decimal number except that it uses base 2 for exponent.

- For eg, the binary number +1001.11 is represented as 8-bit fraction and 6-bit exponent as follows:

Fraction	Exponent
01001110	000100

- The above fraction has 0 in the left most position to denote positive.
- The above floating point number is equivalent to

$$m \times 2^{e} \rightarrow +(.1001110)_2 \times 2^{+4}$$

- A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero.
- For example, the decimal number 350 is normalized but 00035 is not.
- Another example, the 8-bit binary number 00011010 is not normalized because of 3 leading 0's.
- A floating-point number that has a zero in the most significant bit position of the mantissa is said to have an underflow.
- To normalize the number that contains an underflow, it is necessary to shift the mantissa to the left and decrement the exponent until a non-zero appear in the first position.

4.5.1 Floating-point Arithmetic operations

- Arithmetic operations with floating-point numbers are more complicated than with fixed-point numbers and their execution takes longer and requires more complex hardware.
- Adding or subtracting two numbers requires first an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissas.
- Consider the sum of the following floating-point numbers:

$$\mathbf{.5372400 \times 10^{2}}$$
$$\mathbf{+\ .1580000 \times 10^{-1}}$$

- It is necessary that the two exponents be equal before the mantissas can be added. We can either shift the first number three positions to the left, or shift the second number three positions to the right.
- When the mantissas are stored in registers, shifting to the left causes a loss of most significant digits. Shifting to the right causes a loss of least significant digits. The second

method is preferable because it only reduces the accuracy, while the first method may cause an error.

$$.5372400 \times 10^2$$
$$+ .0001580 \times 10^2$$
$$\text{----------}$$
$$.5373980 \times 10^2$$

- When two normalized mantissas are added, the sum may contain an overflow digit. An overflow can be corrected easily by shifting the sum once to the right and incrementing the exponent.
- When two numbers are subtracted, the result may contain most significant zeros as shown in the following example:

$$.56780 \times 10^5$$
$$- .56430 \times 10^5$$
$$\text{\_\_\_\_\_}$$
$$.00350 \times 10^5$$

- A floating-point number that has a 0 in the most significant position of the mantissa is said to have an underflow. To normalize a number that contains an underflow, it is necessary to shift the mantissa to the left and decrement the exponent until a nonzero digit appears in the first position.
- Floating-point multiplication and division do not require an alignment of the mantissas. The product can be formed by multiplying the two mantissas and adding the exponents.
- Division is accomplished by dividing the mantissas and subtracting the exponents.
- The exponent may be represented in any one of the three representations: signed-magnitude, signed-2's complement, or signed-1's complement. 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 is a positive number that is added to each exponent as the floating-point number is formed, so that internally all exponents are positive.
- Each register is subdivided into two parts.
- The mantissa part has the same uppercase letter symbols as in fixed-point representation.
- The exponent part uses the corresponding lowercase letter symbol.

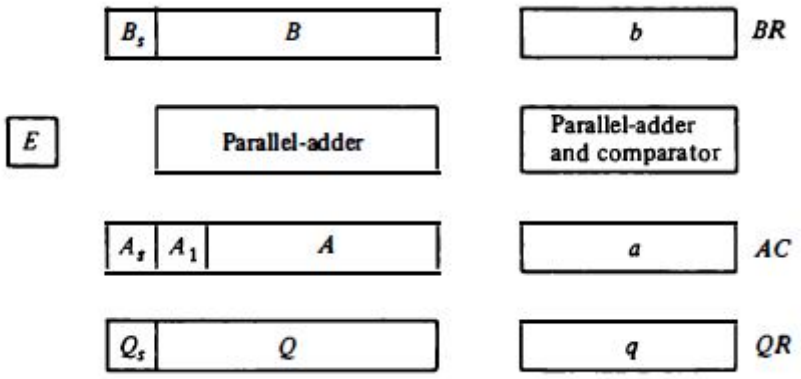


Figure 13: Registers for floating-point arithmetic operations

- A parallel-adder adds the two mantissas and transfers the sum into A and the carry into E.
- A separate parallel-adder is used for the exponents.
- The exponents are also connected to a magnitude comparator that provides three binary outputs to indicate their relative magnitude.
- The number in the mantissa will be taken as a fraction, so the binary point is assumed to reside to the left of the magnitude part.
- The numbers in the registers are assumed to be initially normalized. After each arithmetic operation, the result will be normalized.

4.5.2 Addition or subtraction of two floating-point numbers:

The algorithm can be divided into four consecutive parts:

1. Check for zeros.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result**.**

- The normalization procedure ensures that the result is normalized prior to its transfer to memory. The flowchart for adding or subtracting two floating-point binary numbers is shown in Fig. below.
- If BR is equal to zero, the operation is terminated, with the value in the AC being the result. If AC is equal to zero, we transfer the content of BR into AC and also complement its sign if the numbers are to be subtracted.

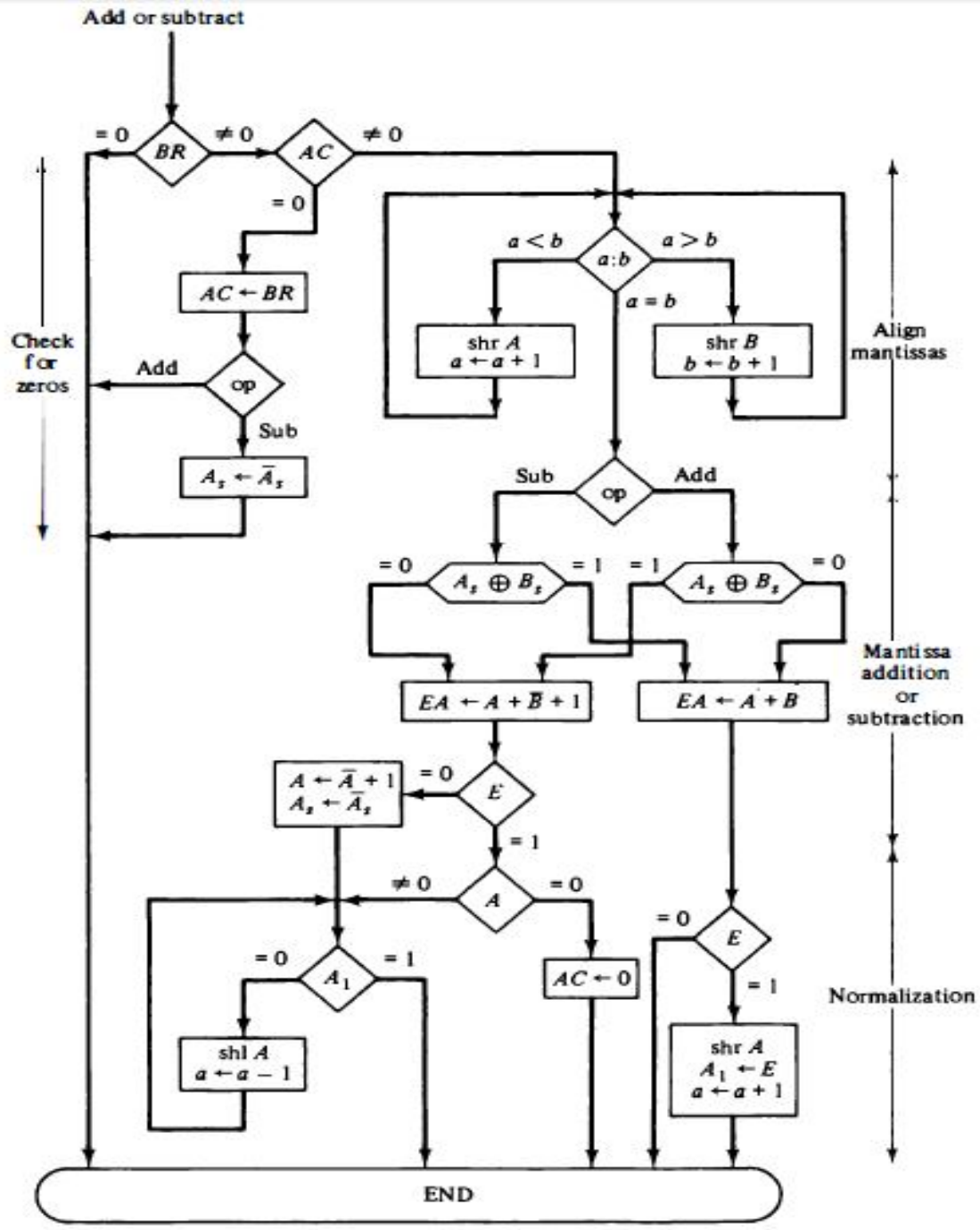


Figure 14: Addition and subtraction of floating-point numbers

- The magnitude comparator attached to exponents a and b provides three outputs that indicate their relative magnitude.
  - i. If the two exponents are equal, we go to perform the arithmetic operation.
  - ii. If the exponents are not equal, the mantissa having the smaller exponent is shifted to the right and its exponent incremented.

This process is repeated until the two exponents are equal.

- The magnitude part is added or subtracted depending on the operation and the signs of the two mantissas. If an overflow occurs when the magnitudes are added, it is transferred into flip-flop E. If E is equal to 1, the bit is transferred into A_1 and all other bits of A are shifted right.
- The exponent must be incremented to maintain the correct number. No underflow may occur in this case because the original mantissa that was not shifted during the alignment was already in a normalized position.
- If the magnitudes were subtracted, the result may be zero or may have an underflow. If the mantissa is zero, the entire floating-point number in the AC is made zero. Otherwise, the mantissa must have at least one bit that is equal to 1.
- The mantissa has an underflow if the most significant bit in position A_1 is 0. In that case, the mantissa is shifted left and the exponent decremented. The bit in A_1 is checked again and the process is repeated until it is equal to 1. When A_1= 1, the mantissa is normalized and the operation is completed.

4.5.3 Multiplication

- The multiplication of two floating-point numbers requires that we multiply the mantissas and add the exponents. No comparison of exponents or alignment of mantissas is necessary.
- The multiplication algorithm can be subdivided into four parts:
  1. Check for zeros.
  2. Add the exponents.
  3. Multiply the mantissas.
  4. Normalize the product.
- Steps 2 and 3 can be done simultaneously if separate adders are available for the mantissas and exponents.
- The two operands are checked to determine if they contain a zero. If either operand is equal to zero, the product in the AC is set to zero and the operation is terminated.

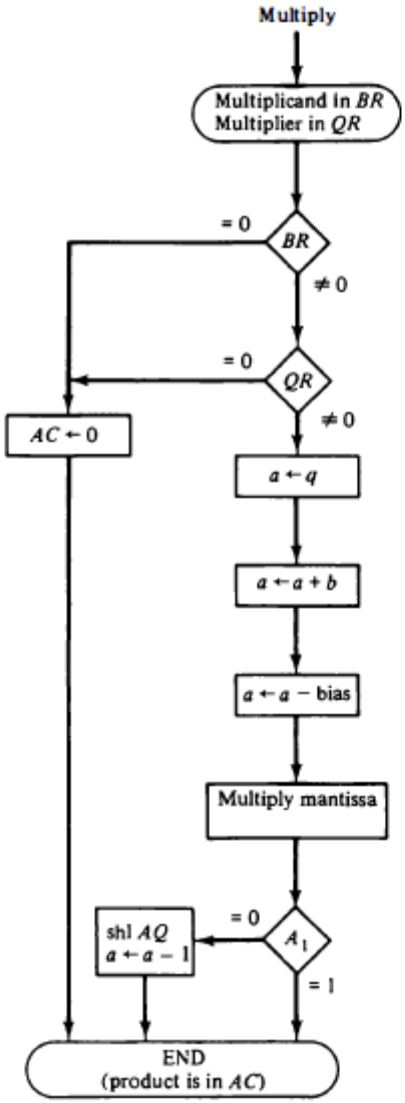


Figure 15: Multiplication of floating-point numbers

- If neither of the operands is equal to zero, the process continues with the exponent addition. The exponent of the multiplier is in q and the adder is between exponents a and b. It is necessary to transfer the exponents from q to a, add the two exponents, and transfer the sum into a.
- Since both exponents are biased by the addition of a constant, the exponent sum will have double this bias. The correct biased exponent for the product is obtained by subtracting the bias number from the sum.
- The multiplication of the mantissas is done as in the fixed-point case with the product residing in A and Q.

- The product may have an underflow, so the most significant bit in A is checked. If it is a 1, the product is already normalized. If it is a 0, the mantissa in AQ is shifted left and the exponent decremented.
- Note that only one normalization shift is necessary. The multiplier and multiplicand were originally normalized and contained fractions. The smallest normalized operand is 0.1, so the smallest possible product is 0.01. Therefore, only one leading zero may occur.

4.5.4 Division

- Floating-point division requires that the exponents be subtracted and the mantissas divided.
- The mantissa division is done as in fixed-point except that the dividend has a single-precision mantissa that is placed in the AC.
- The check for divide-overflow is the same as in fixed-point representation. However, with floating-point numbers the divide-overflow imposes no problems.
- If the dividend is greater than or equal to the divisor, the dividend fraction is shifted to the right and its exponent incremented by 1.
- For normalized operands this is a sufficient operation to ensure that no mantissa overflow will occur. The operation above is referred to as a dividend alignment.
- The division of two normalized floating-point numbers will always result in a normalized quotient provided that a dividend alignment is carried out before the division. Therefore, unlike the other operations, the quotient obtained after the division does not require normalization. The division algorithm can be subdivided into five parts:
  1. Check for zeros.
  2. Initialize registers and evaluates the sign.
  3. Align the dividend.
  4. Subtract the exponents.
  5. Divide the mantissas.
- The two operands are checked for zero. If the divisor is zero, it indicates an attempt to divide by zero, which is an illegal operation. The operation is terminated with an error message.

- An alternative procedure would be to set the quotient in QR to the most positive number possible (if the dividend is positive) or to the most negative possible (if the dividend is negative).If the dividend in AC is zero, the quotient in QR is made zero and the operation terminates.
- If the operands are not zero, we proceed to determine the sign of the quotient and store it in Q_s. The sign of the dividend in A_s is left unchanged to be the sign of the remainder.
- Next, the divisor exponent is subtracted from the dividend exponent. Since both exponents were originally biased, the subtraction operation gives the difference without the bias.

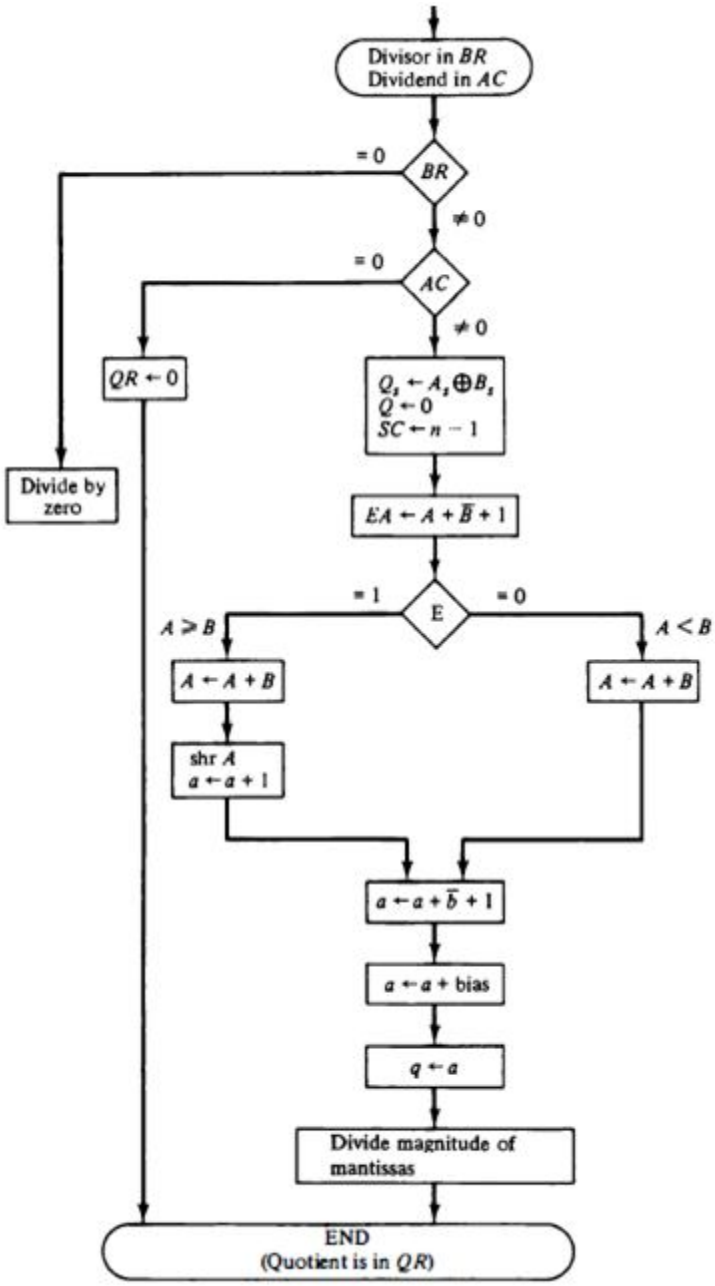


Figure 16: Division of floating point numbers

- The bias is then added and the result transferred into q because the quotient is formed in QR. The magnitudes of the mantissas are divided as in the fixed-point case. After the operation, the mantissa quotient resides in Q and the remainder in A.

Unit - IV

Assignment-Cum-Tutorial Questions

Section - A

1. 11110011 in signed 2's complement system is_____? [ ]
   a. -13 b. +13 c. +14 d. -14
2. The 2's complement of binary number 10000000 is? [ ]
   a. 00000000 c.11111111
   b. 01111111 d.10000000
3. The content of DR register is 0001010111001101 and AC register is 0010010111001101.After the execution of following microoperation AC←AC+DR the result in AC is? [ ]
   a. 0011101110011010 c.1011100011011011
   b. 1011101111011011 d.1011101001000011
4. In computers subtraction is carried out generally by_____? [ ]
   a. 1's complement method c. 2's complement method
   b. Signed magnitude method d. BCD subtraction method
5. For the example 0110x1001the first and second numbers are called as_____? [ ]
   a. Multiplier, Multiplicand c. Multiplier,Multiplier
   b. Multiplicand,Multiplier d. Multiplicand, Multiplicand
6. Booth algorithm is for_____? [ ]
   a. Addition b. subtraction c. division d. multiplication
7. For the example1110+1001,the first and second numbers are called as_____? [ ]
   i. Augend, addend c. addend ,augend
   ii. Addend, addend d. augend, augend
8. For the example1110-1001, the first and second numbers are called as_____? [ ]
   a. minuend, subrahend c. addend, augend
   b. minuend, minued d. augend, subtrahend
9. In Booth multiplication algorithm, which logical microoperation is performed on the partial product? [ ]
   a. Circular shift left c. Arithmetic shift right
   b. Circular shift right d. Arithmetic shift left
10. Perform the subtraction 1011100-1010100 using signed magnitude subtraction algorithm?
    a. 1001000 c. 1001110 [ ]
    b. 1001100 d. 1001111
11. A floating point number that has a 0 in MSB of mantissa is said to have? [ ]

a. Underflow c. Important number

b. Overflow d. Undefined

12. Normalization is performed in _____ [ ]

a. Floating point representation c. Fixed point representation

b. Signed-magnitude representation d. signed-2's complement

13. For comparing the signs of signed magnitude numbers one of the following logic gates is used? [ ]

a. EX-OR b. OR c. AND d. NOT

14. The decimal integer value of 11011001 in signed magnitude representation is__? [ ]

a. -98 b. -89 c. +98 d. +89

15. When the following signed magnitude numbers are added, give the value of AVF after the addition :1011111,1101101 [ ]

a. 10 b. 01 c.0 d. 1

16. Arithmetic shift left operation _____ [ ]

a. Produces same result as obtained with logical shift left operation

b. Causes the sign bit to remain always unchanged

c. Needs additional hardware to preserve the sign bi

d. Is not applicable for signed 2's complement representation

17. If the magnitude of a result exceeds the largest finite number that can be represented in the format of the operation, then it is called_____? [ ]

a. Carry b. Underflow c. Overflow d. Both a and b

18. The floating point binary number is given as 01001110 as the fraction part and 000100 as 6-bit exponent. The number in decimal is? [ ]

a. -6133.789 b. -6131.789 c.-6132.789 d.+6132.789

19. The 1's complement of a binary number is obtained by subtracting its magnitude from (where n is the number of bits used to store the integer in binary)? [ ]

a. 2^n b. 2^n-1 c. 2^n+1 d. 2^n-1

20. When we perform subtraction of -7 and 1 the answer in 2's complement is? [ ]

a. 1001 b. 11000 c. 1000 d. 1110

21. Which representation is most efficient to perform arithmetic operations on the numbers?

a. 1's complement c. 2's complement [ ]

b. Signed 2's complement d. Signed Magnitude

Section - B

1. Explain addition/subtraction operations of fixed point representation with the help of a flowchart.
2. With an example, explain procedure for 2's Complement Subtraction.
3. Illustrate Hardware configuration for Signed Magnitude addition and subtraction of two fixed point numbers.
4. Write algorithm for Multiplication of Signed Magnitude data (Flowchart for multiply operation on Signed Magnitude data).
5. Perform the following Arithmetic operations using ***Signed Magnitude representation*** and verify whether there is *Overflow or not*?

   i. (+13) + (+9) **ii.** (+9) + (–13) **iii.** (+10) + (+18) **iv.**(-14) + (-9)

   v. (+18) – (–10) **vi.**(+18) – (+10) **vii.** (-14) – (-9) **viii.**(-13) – (+6)
6. Draw flowchart for addition and subtraction operation on Signed Magnitude representation of data (Algorithm for add and subtract operations on Signed Magnitude representation of data).
7. Apply Booth's algorithm to multiply the numbers 23 and 19 for no. of bits n=6 in each number.
8. Apply Booth's algorithm to multiply the numbers -15 and 20.
9. Perform the signed 2'scomplementmultiplicationfortheoperands:(-22)*(-9).
10. Write Booth algorithm for multiplication of signed 2's complement numbers.
11. What is the 2's complement of the number: -5/8
12. Normalize the number 0.00530×10^5

Section - C

1. The subtraction of a binary number Y from another binary number X, done by adding 2's compliment of Y to X, results in a binary number without overflow. This implies that the result is?

   (GATE 1987) [ ]

   a. Negative and is in normal form

   b. Negative an is in 2's compliment form

   c. Positive and is in normal form

   d. Positive and is in 2's compliment form
2. 2's complement representation of a 16 bit number (one sign bit and 15 magnitude bits) is FFFF. Its magnitude in decimal representation is? **(GATE 1997)** [ ]

   a. 0 b. 1 c. 32,767 d. 65,535
3. An equivalent 2's complement representation of the 2's complement number is 1101 is?

   (GATE 1998) [ ]

   a. 110100 b. 001101 c. 110111 d. 111101

4. The 2's complement representation of – 17 is? **(GATE 2001)** [ ]

   a. 01110 b. 101111 c. 11110 d. 10001

5. The range of signed decimal numbers that can be represented by 6 bit 1'scomplement form is?

   (GATE 2004) [ ]

   a. 31to +31 b. 63to +64 c. 64to +63 d. 32to +31

6. X = 01110 and Y = 11001 are two 5 bit binary numbers represented in 2's complement format. The sum of X and Y represented in 2's compliment format using 6 bits is?

   (GATE 2007) [ ]

   a. 100111 b. 001000 c. 000111 d. 101001

Unit - V

INPUT-OUTPUT ORGANIZATION

Objective:

- To familiarize the working of different I/O devices connected to the system and their inter communication.

Syllabus:

INPUT-OUTPUT ORGANIZATION: Peripheral Devices, input-output interface, asynchronous data transfer, modes of transfer- programmed I/O, priority interrupt, direct memory access, Input –Output Processor (IOP).

Learning Outcomes:

At the end of the unit student will be able to:

1. Summarize different ways in which data transfer takes place between I/O devices and Processor.
2. Differentiate between synchronous and asynchronous communication.

Learning Material

5.1 PERIPHERAL DEVICES

- The input-output subsystem of a computer, referred to as I/O.
- Input or output devices attached to the computer are also called peripherals.
- Among the most common peripherals are keyboards, display units, and printers.
- Devices that are under the direct control of the computer are said to be connected on-line.
- Video monitors are the most commonly used peripherals. They consist of a keyboard as the input device and a display unit as the output device.

5.2 INPUT-OUTPUT INTERFACE

- Input-output interface provides a method for transferring information between internal storage and external I/0 devices.
- Peripherals connected to a computer need special communication links for interfacing them with the CPU.
- The purpose of the communication link is to resolve the differences that exist between the CPU and each peripheral.
- The major differences are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other.

To resolve the above differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called interface units.

5.2.1 I/O Bus and Interface Modules

- Here the I/O bus consists of data lines, address lines, and control lines.
- Each peripheral device has associated with it an interface unit.
- Each interface decodes the address and control received from the I/O bus, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the data transfer between peripheral and processor.

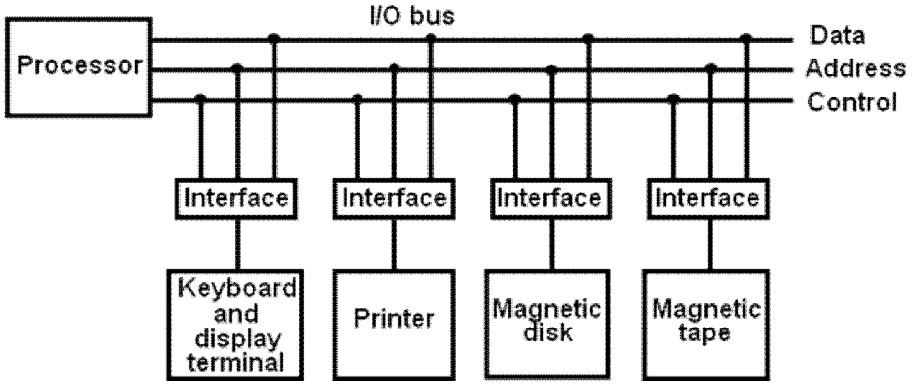


Figure 1: Connection of I/O bus to input output devices

- The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines.
- Each interface attached to the 1/0 bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls.
- There are four types of commands that an interface may receive. They are classified as:
    1. Control command
    2. Status command
    3. Data output command

4. Data input command

- A control command is issued to activate the peripheral and to inform it what to do.
- A status command is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated
- A data output command causes the interface to respond by transferring data from the bus into one of its registers.
- The data input command causes the interface receive an item of data from the peripheral and places it in its buffer register.

5.2.2 I/O Bus versus Memory Bus

- In addition to communicating with I/O, the processor must communicate with the memory unit.
- Like the I/O bus, the memory bus contains data, address, and read/write control lines.
- There are three ways that computer buses can be used to communicate with memory and I/O
  1. Use two separate buses, one for memory and the other for I/O.
  2. Use one common bus for both memory and I/O but have separate control lines for each.
  3. Use one common bus for memory and I/O with common control lines.
- In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O.

5.2.3 Isolated I/O versus Memory-Mapped I/O

Isolated I/O:

- The distinction between a memory transfer and I/O transfer is made through separate read and write lines.
- The I/O read and I/O write control lines are enabled during an I/O transfer.
- The memory read and memory write control lines are enabled during a memory transfer.

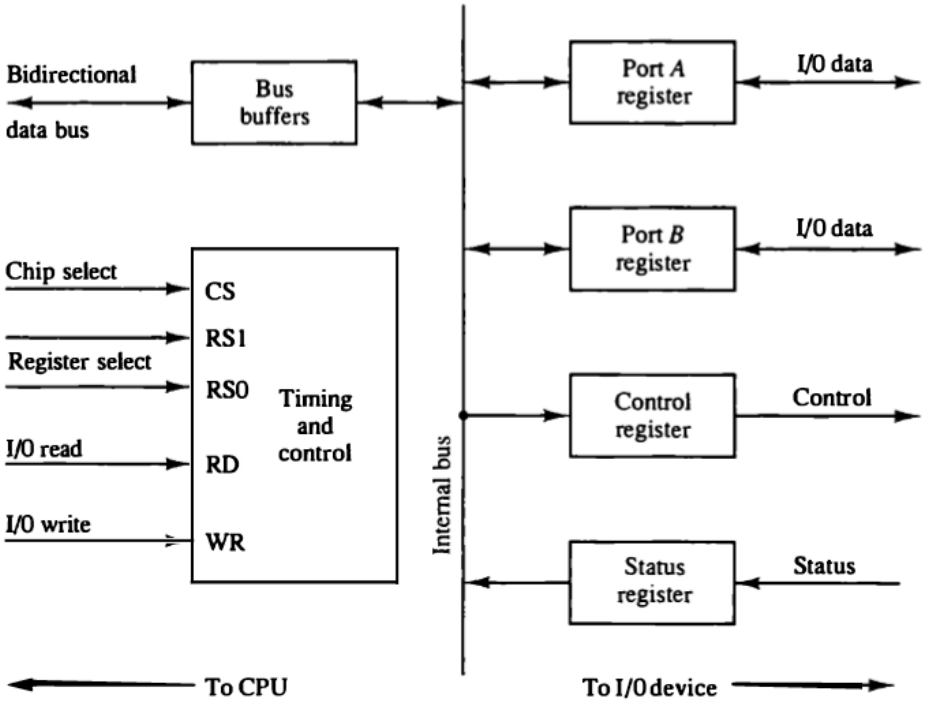
Memory-Mapped I/O:

- It uses the same address space for both memory and I/O.
- In this is the case computers has only one set of read and write signals and do not distinguish between memory and I/O addresses. This configuration is referred to as memory-mapped I/O

5.2.4 Example of I/O Interface

- An example of an I/O interface unit is shown in block diagram form in figure 2. It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits.
- The interface communicates with the CPU through the bidirectional data bus.

- The chip select and register select inputs determine the address assigned to the interface.
- The I/O read and write are two control lines that specify an input or output, respectively.
- The four registers communicate directly with the I/O device attached to the interface.
- The I/O data to and from the device can be transferred into either port A or port B.
- The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of operating modes.
- This circuit gets enabled when the chip select (CS) line is active. The CS is active when the address generated by the processor is for this device.



CS	RS1	RS0	Register selected
0	×	×	None: data bus in high-impedance
1	0	0	Port *A* register
1	0	1	Port *B* register
1	1	0	Control register
1	1	1	Status register

Figure 2: Example of I/O interface unit

5.3. ASYNCHRONOUS DATA TRANSFER

- The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator.
- Two units, such as a CPU and an I/O interface, are designed independent of each other. If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be *synchronous*.
- In most cases, the internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. In that case, the two units are said to be *asynchronous* to each other.
- Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.
- Two methods are available:
  1. Strobe Control
  2. Handshaking

5.3.1. Strobe Control

- A strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
- The strobe control method of asynchronous data transfer employs a single control line.
- The strobe may be activated by either the source or the destination unit.

5.3.1.1 Source initiated strobe

- The following figure 3(a) shows a source-initiated transfer.
- The data bus carries the binary information from source unit to the destination unit. The strobe informs the destination unit when a valid data word is available in the bus.

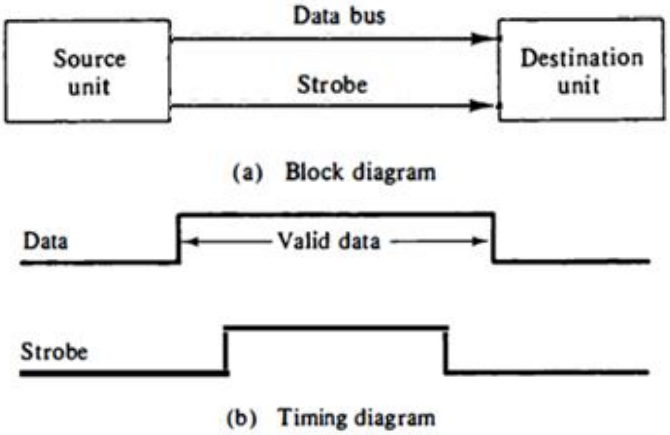


Figure 3: Source initiated strobe for data transfer

- The timing diagram is as shown in the above figure 3(b), the source unit first places the data on the data bus. After a brief delay the source activates the strobe pulse.
- The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- Often, the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers.
- The source removes the data from the bus after a brief period by disabling its strobe pulse.
- Actually, the source does not have to change the information in the data bus. The fact that the strobe signal is disabled indicates that the data bus does not contain valid data.

5.3.1.2 Destination initiated strobe

- The following figure 4 shows a data transfer initiated by the destination unit. In this case the destination unit activates the strobe pulse, informing the source to provide the data.
- The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus longenough for the destination unit to accept it.
- The falling edge of the strobe pulse can be used again to trigger a destination register. The destination unit then disables the strobe.
- The source removes the data from the bus after a predetermined time interval.

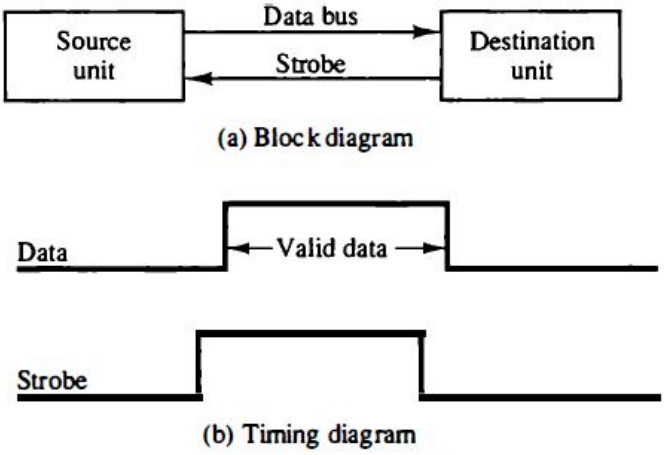


Figure 4: Destination initiated strobe for data transfer

5.3.2 Handshaking

- The disadvantage of the strobe method is that the source unit that initiates the transfer has no knowledge of the destination unit has actually received the data item or not from the bus.
- Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.

- The handshake method solves these problems by introducing a second control signal that provides a reply to the unit that initiates the transfer.
- The basic principle of the two-way handshaking method of data transfer is as follows:
    - One control line is in the same direction as the data flow i.e., from the source to the destination. It is used by the source to inform the destination about the data validity.
    - The other control line is in the other direction i.e., from the destination to the source. It is used by the destination unit to inform the source regarding data acceptance.
    - The sequence of control signals exchanged during the transfer depends on the unit that initiates the transfer.

5.3.2.1 Source initiated transfer using handshaking

- The following figure 5 shows the data transfer procedure when initiated by the source.
- The two handshaking lines are *data valid*, which is generated by the source unit, and *data accepted*, generated by the destination unit.
- The timing diagram shows the exchange of signals between the two units. The sequence of events listed in part (c) shows the four possible states that the system can be at any given time.

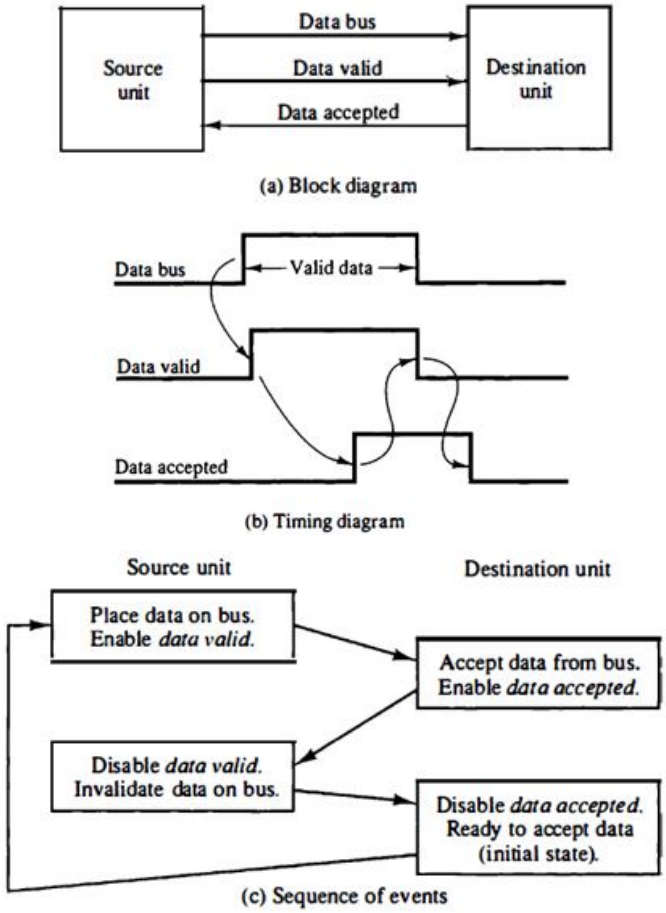


Figure 5: Source initiated transfer using handshaking

5.3.2.2 Destination initiated transfer using handshaking

- The destination-initiated transfer using handshaking is shown in the following figure 6.
- The source unit in this case does not place data on the bus till it receives the ready for data signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source-initiated case.
- The only difference between the source-initiated and the destination-initiated transfer is in their choice of initial state.

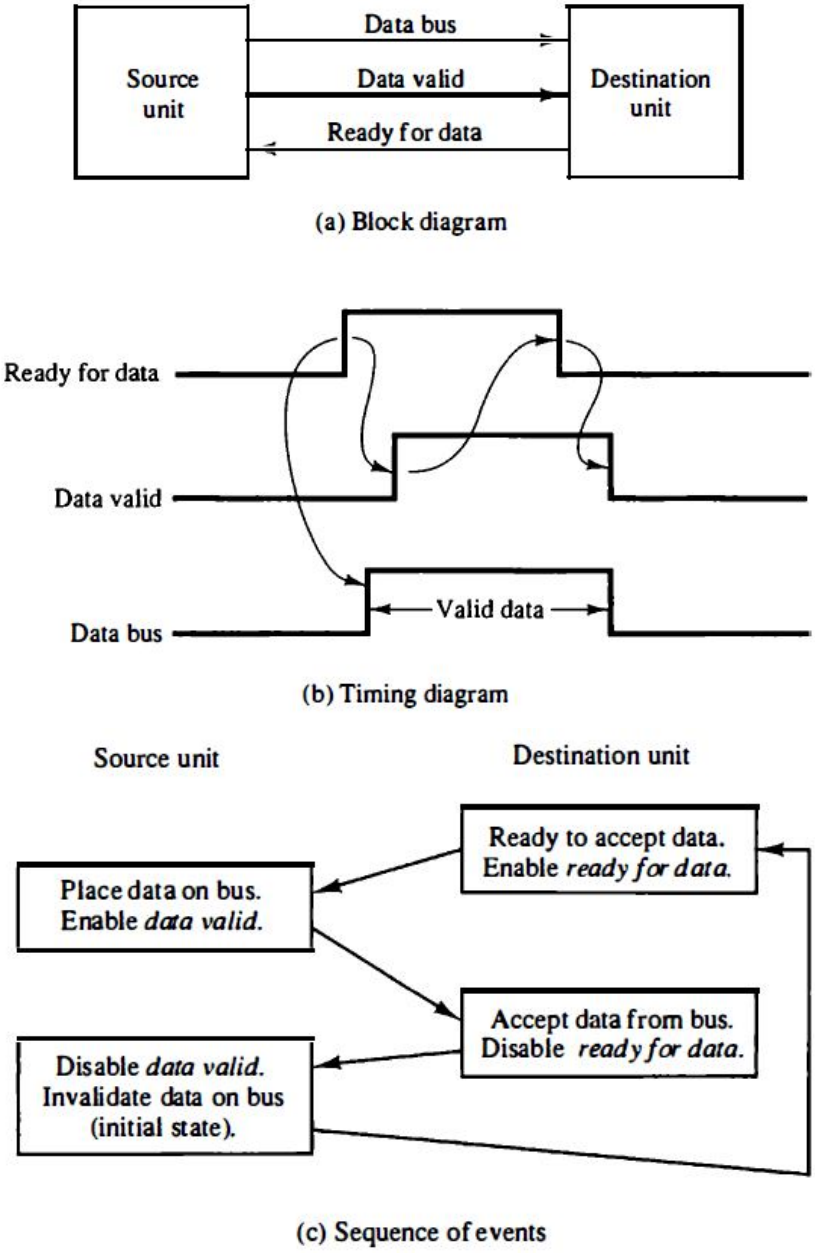
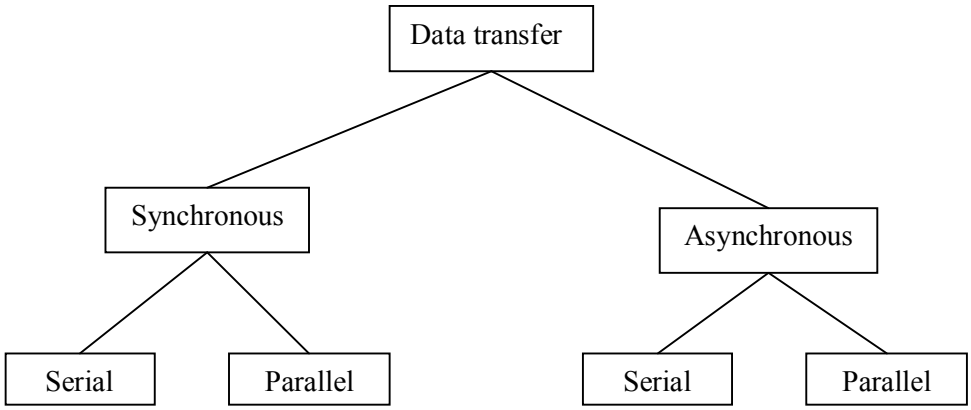


Figure 6: Destination initiated transfer using handshaking



5.3.3 Asynchronous Serial Transfer

- The transfer of data between two units may be done in parallel or serial.
- In parallel data transmission, each bit of the message has its own path and the total message is transmitted at the same time. This means that an n-bit message must be transmitted through n separate conductor paths.
- In serial data transmission,each bit in the message is sent in sequence one at a time. This method requires the use of one pair of conductors or one conductor and a common ground.
- Parallel transmission is faster but requires many wires. It is used for short distances and where speed is important.
- Serial transmission is slower but is less expensive since it requires only one pair of conductors.
- A serial asynchronous data transmission technique used in many interactive terminals employs special bits that are inserted at both ends of the character code. With this technique, each character consists of three parts: a *start bit*, the *character bits*, and *stop bits*.
- The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character.
- The last bit called the stop bit is always a 1.

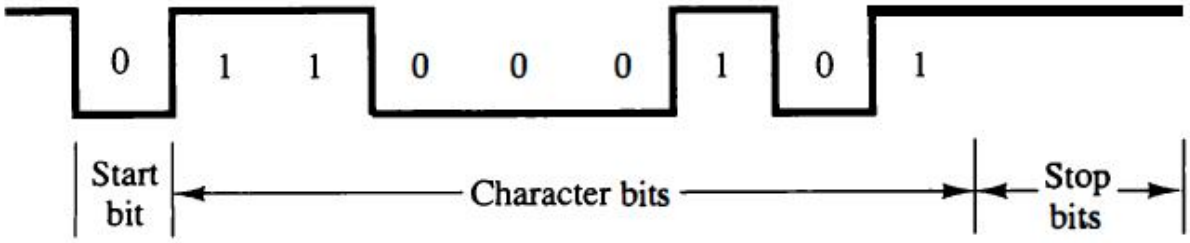


Figure 7: Asynchronous serial transmission

- A transmitted character can be detected by the receiver from knowledge of the transmission rules:

  1. When a character is not being sent, the line is kept in the 1-state.

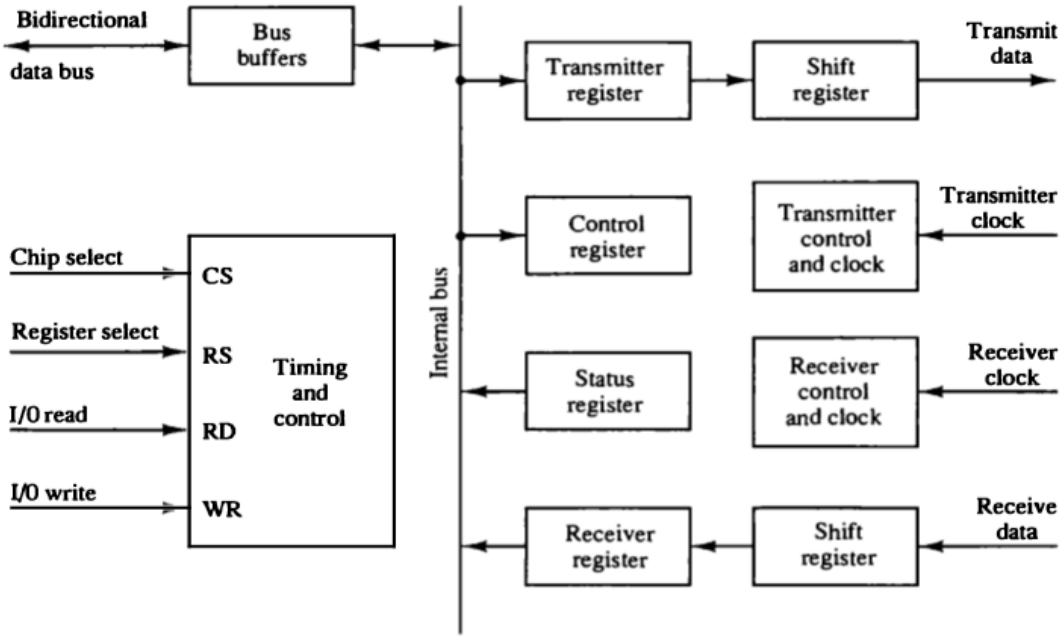
  2. The initiation of a character transmission is detected from the start bit, which is always 0.

  3. The character bits always follow the start bit.

  4. After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time.
- Using these rules, the receiver can detect the start bit when the line goes from 1 to 0.
- After the character bits one or two *stop bits* are transmitted.
- At the end of the character the line is held at the 1-state for a period of at least one or two bit times so that both the transmitter and receiver can resynchronize.
- The interface between computer and similar interactive terminals is called an asynchronous communication interface or a universal asynchronous receiver transmitter (UART).

5.3.4 Asynchronous Communication Interface

- The block diagram of an asynchronous communication interface is shown in Fig 8. It functions as both a transmitter and a receiver. The interface is initialized for a particular mode of transfer by means of a control byte that is loaded into its control register.
- The transmitter register accepts a data byte from the CPU through the data bus. This byte is transferred to a shift register for serial transmission.
- The receiver portion receives serial information into another shift register, and when a complete data byte is accumulated, it is transferred to the receiver register.
- The CPU can select the receiver register to read the byte through the data bus.
- The bits in the status register are used for input and output flags and for recording certain errors that may occur during the transmission.
- The CPU can read the status register to check the status of the flag bits and to determine if any errors have occurred.
- The chip select (CS) input is used to select the interface through the address bus. The register select (RS) is associated with the read (RD) and write (WR) controls.
- Two registers are write-only and two are read-only. The register selected is a function of the RS value and the RD and WR status, as listed in the table accompanying the diagram.



CS	RS	Operation	Register selected
0	×	×	None: data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

Figure 8: Block diagram of a typical asynchronous communication interface

- The operation of the transmitter portion of the interface is that the CPU reads the status register and checks the flag to see if the transmitter register is empty.
- If status register is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full.
- The operation of the receiver portion of the interface is similar. The received data input is in the 1-state when the line is idle. The receiver control monitors the receive-data line for a 0 signal to detect the occurrence of a start bit.

5.3.5 First-In, First-Out Buffer

- A first-in, first-out (FIFO) buffer is a memory unit that stores information in such a manner that the item first in is the item first out.
- A FIFO buffer comes with separate input and output terminals.

- The important feature of this buffer is that it can input data and output data at two different rates and the output data are always in the same order in which the data entered the buffer.
- If the source unit is slower than the destination unit, the buffer can be filled with data at a slow rate and later emptied at the higher rate.
- If the source is faster than the destination, the FIFO is useful for those cases where the source data arrive in bursts that fill out the buffer but the time between bursts is long enough for the destination unit to empty some or all the information from the buffer.
- Thus a FIFO buffer can be useful in some applications when data are transferred asynchronously.
- The logic diagram of a typical 4 x 4 FIFO buffer is shown in Fig 9.

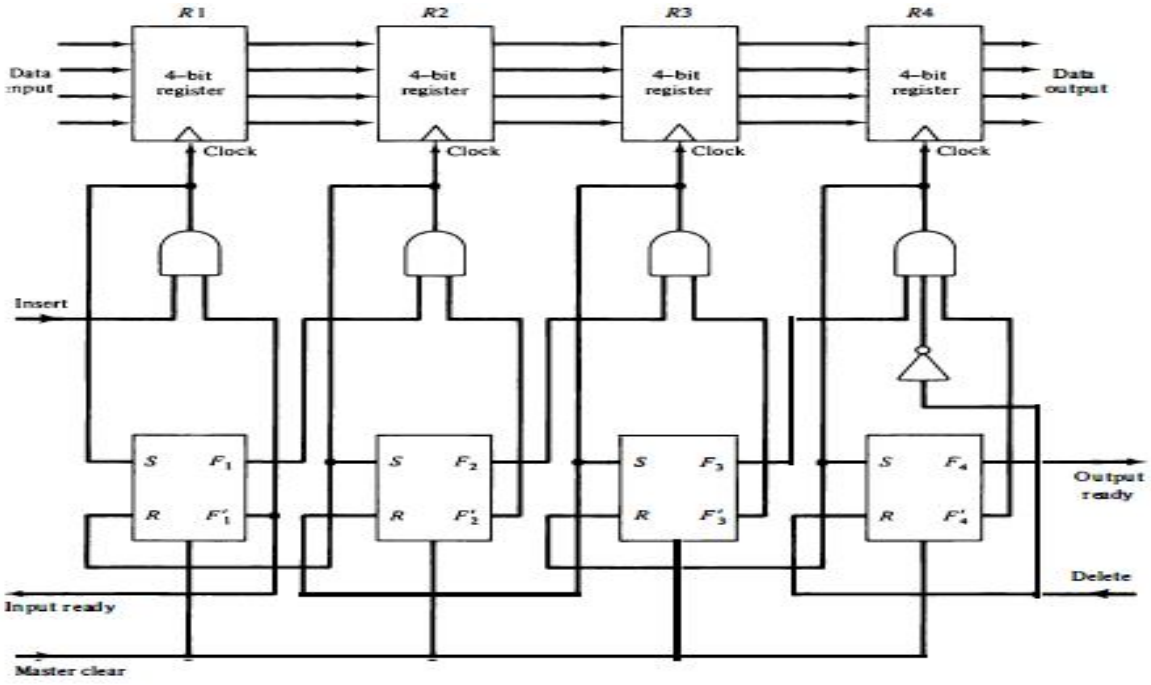


Figure 9: First-In First-Out Buffer

- It consists of four 4-bit registers RI, I = 1, 2, 3, 4, and a control register withflip-flops F_i, i = 1, 2, 3, 4, one for each register.
- A flip-flop F_i in the control register that is set to 1 indicates that a 4-bit data word is stored in the corresponding register RI. A 0 in F_i indicates that the corresponding register does not contain valid data.
- If F_i= 1 and the F_{i+1}'=1 then a clock is generated causing register R_{i+1} to accept the data from register R_i. (i.e. $R_{i+1} \leftarrow R_i$).
- The same clock transition sets F_{i+1}=1 and resets F_i'=1. This causes the control flag to move one position to the right together with the data.

- Data are inserted into the buffer provided that the input ready signal is enabled. This occurs when the first control flip-flop F1 is reset, indicating that register R1 is empty.
- The same clock sets F_1, which disables the input ready control, indicating that the FIFO is now busy and unable to accept more data.
- The ripple-through process begins provided that R2 is empty. The data in R1 are transferred into R2 and F_1 is cleared. This enables the input ready line, indicating that the inputs are now available for another data word.
- If the FIFO is full, F1 remains set and the input ready line stays in the 0 state.
- The output ready control line is enabled when the last control flip-flop F4 is set, indicating that there are valid data in the output register R4.
- The output data from R4 are accepted by a destination unit, which then enables the delete control signal. This resets F_4, causing output ready to disable, indicating that the data on the output are no longer valid.

5.4 MODES OF TRANSFER

- Data transfer between the central computer and 110 devices may be handled in a variety of modes.
- Data transfer to and from peripherals may be handled in one of three possible modes:
  1. Programmed I/O
  2. Interrupt-Initiated I/O
  3. Direct memory access (DMA)

5.4.1. Programmed I/O

- Programmed I/O operations are the result of I/O instructions written in the computer program.
- Each data item transfer is initiated by an instruction in the program.
- In the programmed I/O method, the I/Odevice does not have direct access to memory.
- A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory.
- Other instructions may be needed to verify that the data are available from the device and to count the numbers of words transferred.
- An example of data transfer from an I/O device through an interface into the CPU is shown in Fig. 10.

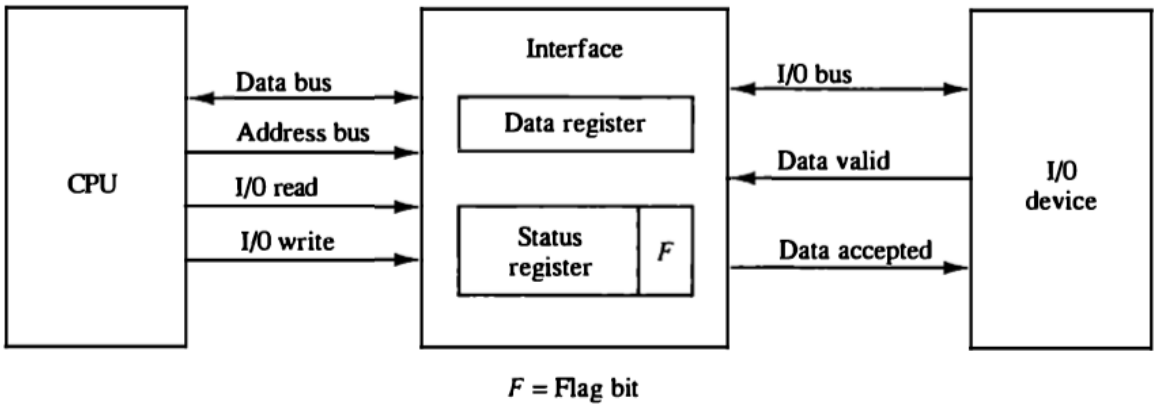


Figure 10: Data transfer from I/O device to CPU

- When a byte of data is available, the device places it on the I/O bus and enables its data valid line.
- The interface accepts the byte into its data register and enables the data accepted line.
- The interface sets a bit in the status register that we will refer to as an For "flag" bit.
- The device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface.
- A flowchart of the program that must be written for the CPU is shown in Fig 11.
- The transfer of each byte requires three instructions:

  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.

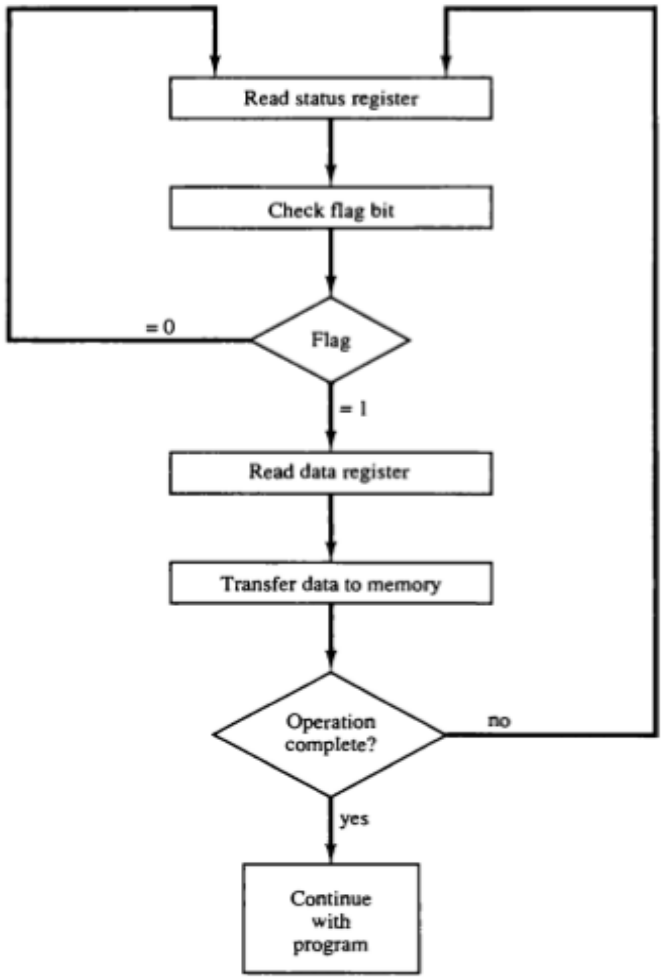


Figure 11: Flowchart for CPU program to input data

- The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously.

5.4.2. Interrupt-initiated I/O

- In the programmed I/O method, the CPU stays in a loop until the I/O unit indicates that it is ready for data transfer. This is a time-consuming process since it keeps the processor busy needlessly.
- It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device.
- In the meantime the CPU can proceed to execute another program.
- The interface meanwhile keeps monitoring the device. When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer.

- Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer, and then returns to the task it was originally performing.
- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.
- The way that the processor chooses the branch address of the service routine varies from one unit to another.
- In principle, there are two methods for accomplishing this. One is called *vectored interrupt* and the other, *non-vectored interrupt*.
- In a *non-vectored interrupt*, the branch address is assigned to a fixed location in memory.
- In a *vectored interrupt*, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector.

5.4.2.1 Priority Interrupt

- A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously.
- Devices with high speed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority.
- When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first.
- Establishing the priority of simultaneous interrupts can be done by software or hardware.
- A **polling** procedure is used to identify the highest-priority source by software means. In this method there is one common branch address for all interrupts.
- The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt.
- The highest-priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise, the next-lower-priority source is tested, and so on.
- The disadvantage of the software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device.
- In this situation a hardware priority-interrupt unit can be used to speed up the operation.
- A hardware priority-interrupt unit accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination.

- To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly.
- Thus no polling is required because all the decisions are established by the hardware priority-interrupt unit.
- The hardware priority function can be established in two ways.
  1. Serial connection (daisy chaining)
  2. Parallel connection

5.4.3 Daisy-Chaining Priority (Serial connection)

- The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt.
- The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain.
- This method of connection between devices and the CPU is shown in Fig. 12.

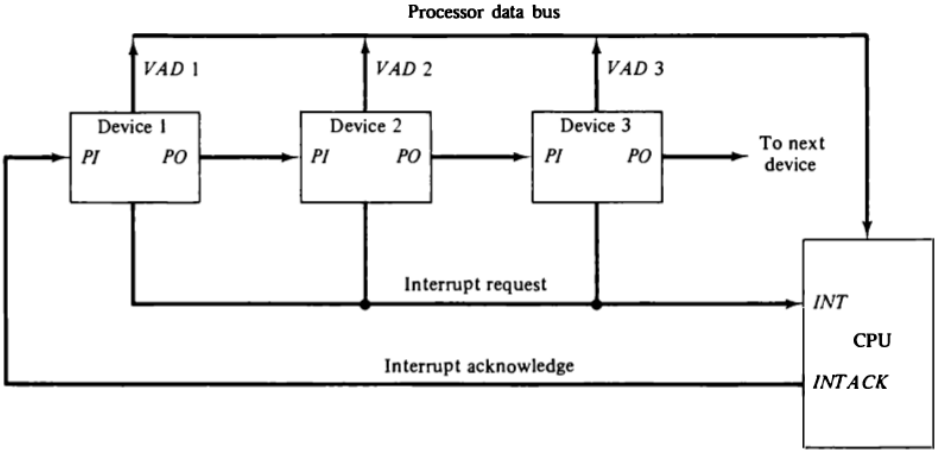


Figure 12: Daisy-Chain Priority interrupt

- The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU.
- When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU.
- The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (priority in) input. The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt.

- If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.
- A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked.
- A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output.
- If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output.
- Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus.
- The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU.
- The following figure 13 shows the internal logic that must be included within each device when connected in the daisy-chaining scheme.

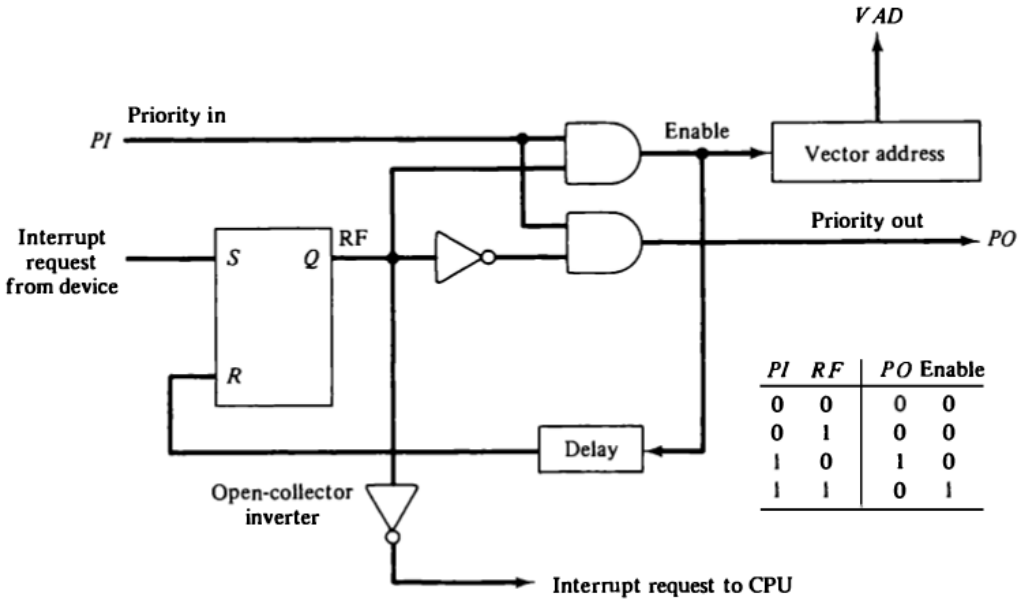


Figure 13: One stage of the daisy-chain priority arrangement

- The device sets its RF flip-flop when it wants to interrupt the CPU.
- If PI = 0, both PO and the enable line to VAD are equal to 0, irrespective of the value of RF.
- If PI = 1 and RF = 0, then PO = 1 and the vector address is disabled. This condition passes the acknowledge signal to the next device through PO.

- The device is active when PI = 1 and RF = 1. This condition places a 0 in PO and enables the vector address for the data bus.
- The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

5.4.4 Parallel Priority Interrupt

- The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device.
- Priority is established according to the position of the bits in the register. In addition to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request.
- The priority logic for a system of four interrupt sources is shown in Fig. 14. It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions.

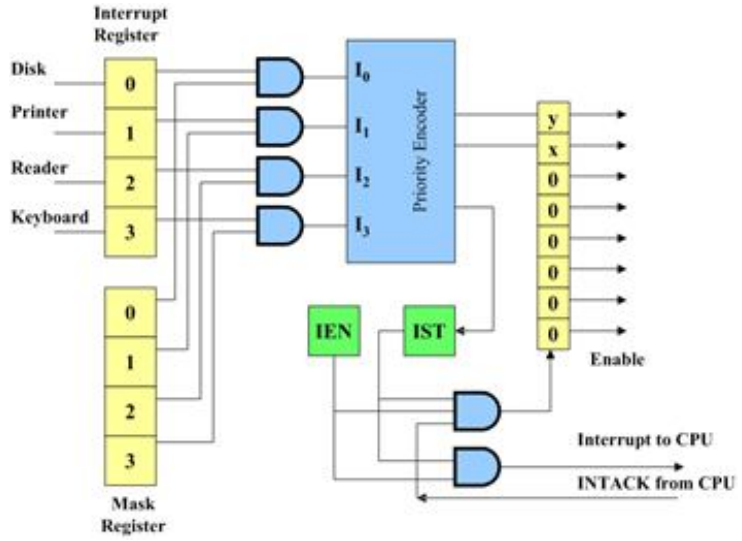


Figure 14: Priority interrupt hardware

- The magnetic disk, being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard.
- The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register.
- Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder.
- In this way an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU.

- Another output from the encoder sets an interrupt status flip-flop IST when an interrupt that is not masked occurs. The interrupt enable flip-flop IEN can be set or cleared by the program to provide an overall control over the interrupt system.
- The outputs of IST ANDed with IEN provide a common interrupt signal for the CPU.
- The interrupt acknowledge INTACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus.

Priority Encoder

- The priority encoder is a circuit that implements the priority function.
- The logic of the priority encoder is such that if two or more inputs arrive at the same time, the input having the highest priority will take precedence.
- The truth table of a four input priority encoder is given in Table 2.

TABLE 2: Priority Encoder Truth Table

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
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	

- The output of the priority encoder is used to form part of the vector address for each interrupt source.

5.5 DIRECT MEMORY ACCESS (DMA)

- The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU.
- Removing the CPU from the path and letting the peripheral device manage the memory buses directly is facilitated by DMA.
- A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.
- The CPU can be allowed to perform its usual operations without any frequent interruptions from I/O devices by using special control signals.
- These control signals (bus request) from the DMA controller requests the CPU to relinquish control of the buses and place them in high impedance state.

- The CPU activates the (bus grant) output to inform the external DMA that the buses are in high impedance state.
- The DMA can now take control of the buses to conduct memory transfers without processors intervention.
- Upon completion, the DMA disables the bus request line following which CPU disables the bus grant, takes control of the buses and returns to its normal operation.
- The DMA can directly communicate with memory after the bus grant is issued by the CPU. Data transfer can be made by transferring data in burst or an alternative technique called cycle stealing.
- Burst transfer is useful while communicating with fast devices and transfer can be done in a continues burst while the DMA controller is the master of the buses.
- In cycle stealing DMA controller transfers one data word at a time, after which it must return control of the buses to the CPU.

5.5.1 DMA Controller

- The DMA controller needs the usual circuits of an interface to communicate with CPU and I/O device. In addition it needs a word count register, and a set of address lines.
- The address register and address lines are used for direct communication with memory. The word count register specifies the number of words that must be transferred. The data transfer may be done directly between the device and memory under control of DMA.

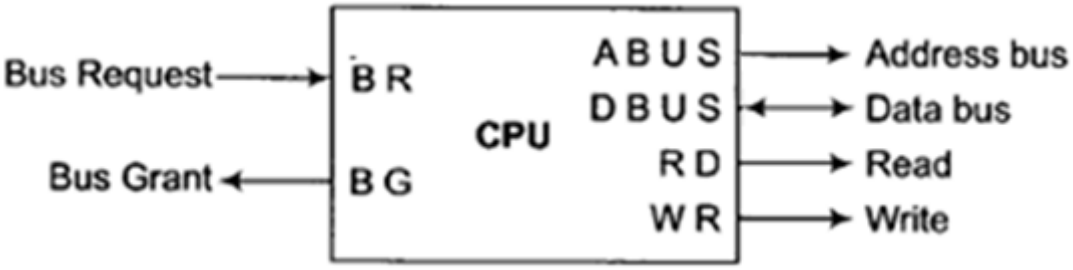


Figure 15: CPU BUS Signals for DMA Transfer

- The unit communicates with CPU via the data bus and control lines. The registers in DMA are selected by the CPU through the address bus by enabling the DS and RS inputs.
- The RD(read) and WR(write) inputs are bidirectional. When BG=1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.
- The DMA communicates with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure.
- The DMA controller has 3 registers: an address register, a word count register, and a control register, and a control register.

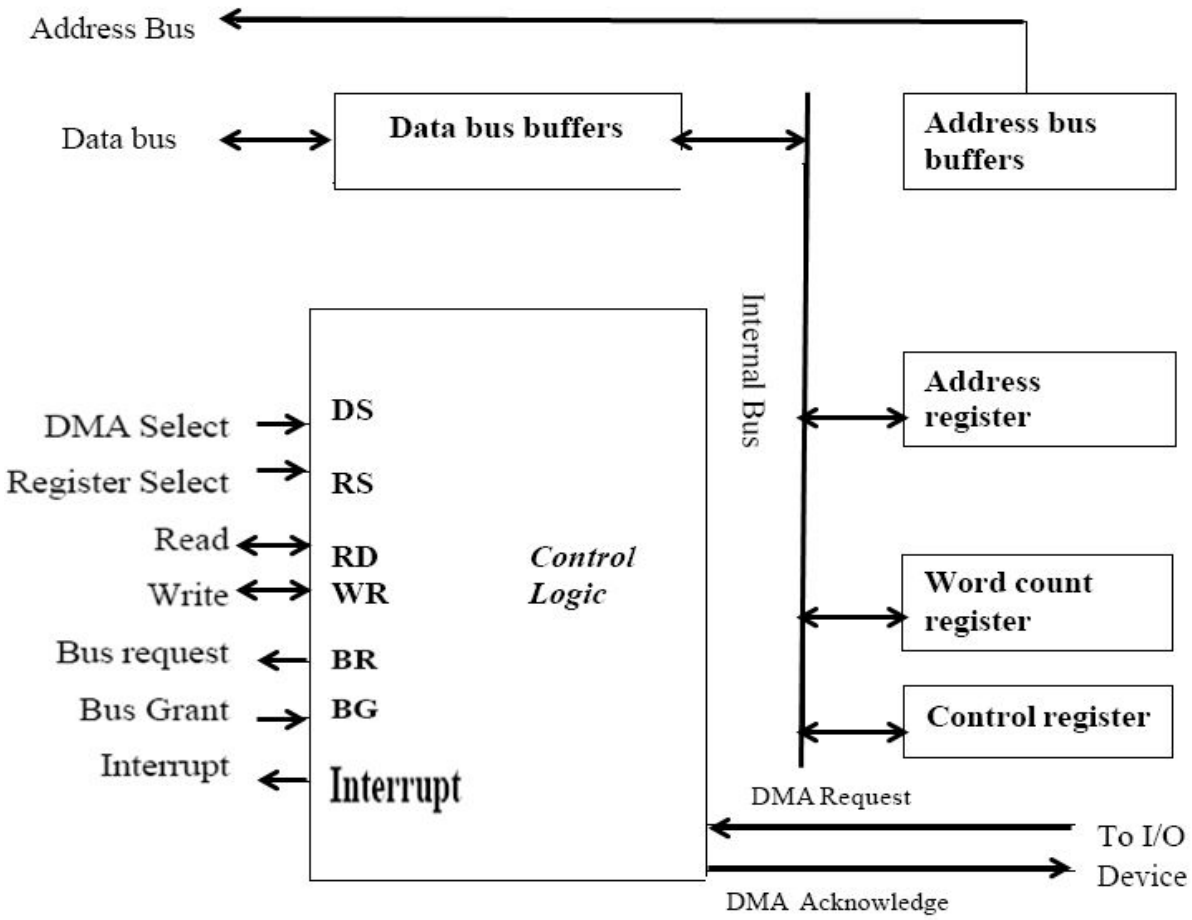


Figure 16: Block diagram for DMA Controller

- The address bits from the address register go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory.
- The word count register holds the number of words to be transferred and is decremented after each transfer.
- The control register specifies the mode of transfer. The CPU can read from or write into the DMA registers via the data bus.
- The DMA is first initialized by the CPU. After that, the DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.
- The initialization process is essentially a program consisting of I/O instructions that include the address for selecting particular DMA registers.
- The CPU initializes the DMA by sending the following information through the data bus:

i. The starting address of the memory block where data are available or where data are to be stored
ii. The word count, which is the number of words in the memory block
iii. Control to specify the mode of transfer such as read and write
iv. A control to start the DMA transfer

5.5.2 DMA Transfer

- The CPU communicates with the DMA through the address and data buses as with any interface unit.
- The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus.
- When the peripheral device sends a DMA request, the DMA controller activates the BR line, requesting the CPU to relinquish the system bus.
- The CPU responds by sending a BG signal informing the DMA that its buses are disabled. The DMA then puts the address in address register on to the address lines initiates the RD or WR signal, and sends DMA acknowledge to the peripheral device.
- When the peripheral device receives a DMA acknowledge, it puts a word in the data bus or receives a word from the data bus.
- The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while CPU is momentarily disabled.

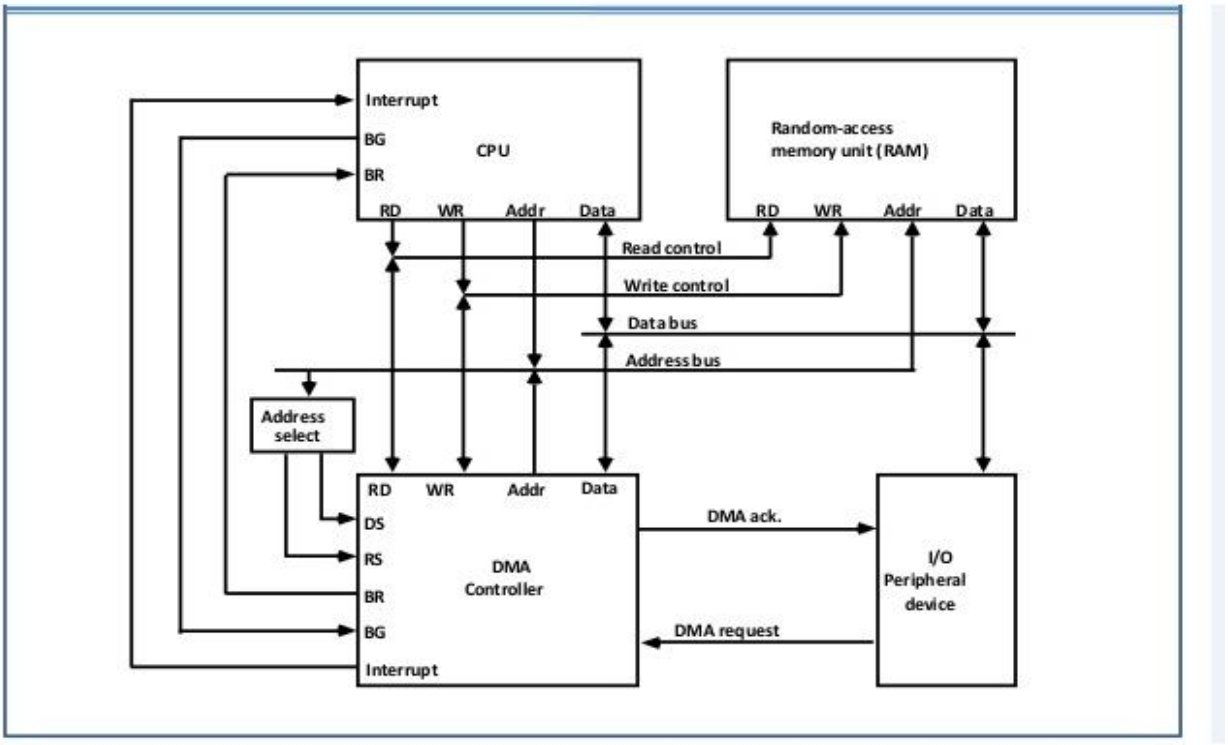


Figure 17: DMA Transfer in a computer system

- For each word that is transferred, the DMA increments its address registers and decrements its word count register.
- If the word count register does not reach zero, the DMA checks the request line coming from the peripheral. When the peripheral requests a transfer, the DMA requests the bus again.
- If the word count register reaches zero, the DMA stops any further transfer and removes its bus request.
- It also informs the CPU of the termination by means of an interrupt. When the CPU responds to the interrupt, it reads the content of the word count register to check for successful transmission.
- DMA transfer is useful in fast transfer of information between magnetic disks and memory.

5.6 INPUT-OUTPUT PROCESSOR (IOP)

- An Input-Output Processor (IOP) may be classified as a processor with direct memory access capability that communicates with all I/O devices.
- Here the computer system can be divided into a memory, and a number of processors comprised of CPU and one or more IOPs.
- The IOP is similar to a CPU except that it is designed to handle the details of the I/O processing.
- The IOP can perform arithmetic, logic, branching and code translation.

- The block diagram of a computer with two processors is shown in Fig. 18. The memory unit occupies a central position and can communicate with each processor by means of DMA.

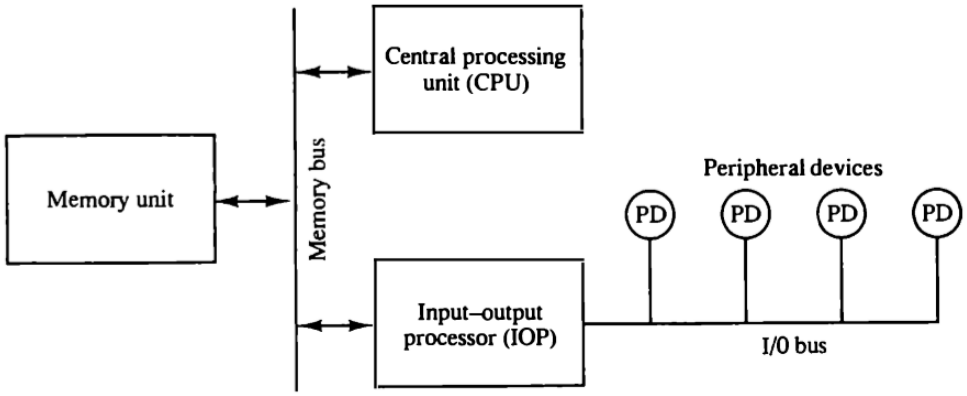


Figure 18: Block diagram of a computer with I/O processor

- The CPU is responsible for processing data needed in computational task.
- The IOP provides a path for transformation of data between various peripheral devices and the memory unit.
- The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources.
- The communication between the IOP and the devices attached to it is similar to the program control method of transfer.
- Communication with the memory is similar to the direct memory access method.
- In most computer systems, the CPU is the master while the IOP is a slave processor. The CPU is assigned the task of initiating all operations, but I/O instructions are executed in the IOP.
- CPU instructions provide operations to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities.
- Instructions that are read from memory by an IOP are sometimes called commands, to distinguish them from instructions that are read by the CPU.

5.6.1 CPU-IOP Communication

- The communication between CPU and IOP may take different forms, depending on the particular computer considered.
- In most cases the memory unit acts as a message center where each processor leaves information for the other.
- The sequence of operations may be carried out as shown in the flowchart of Fig. 19.

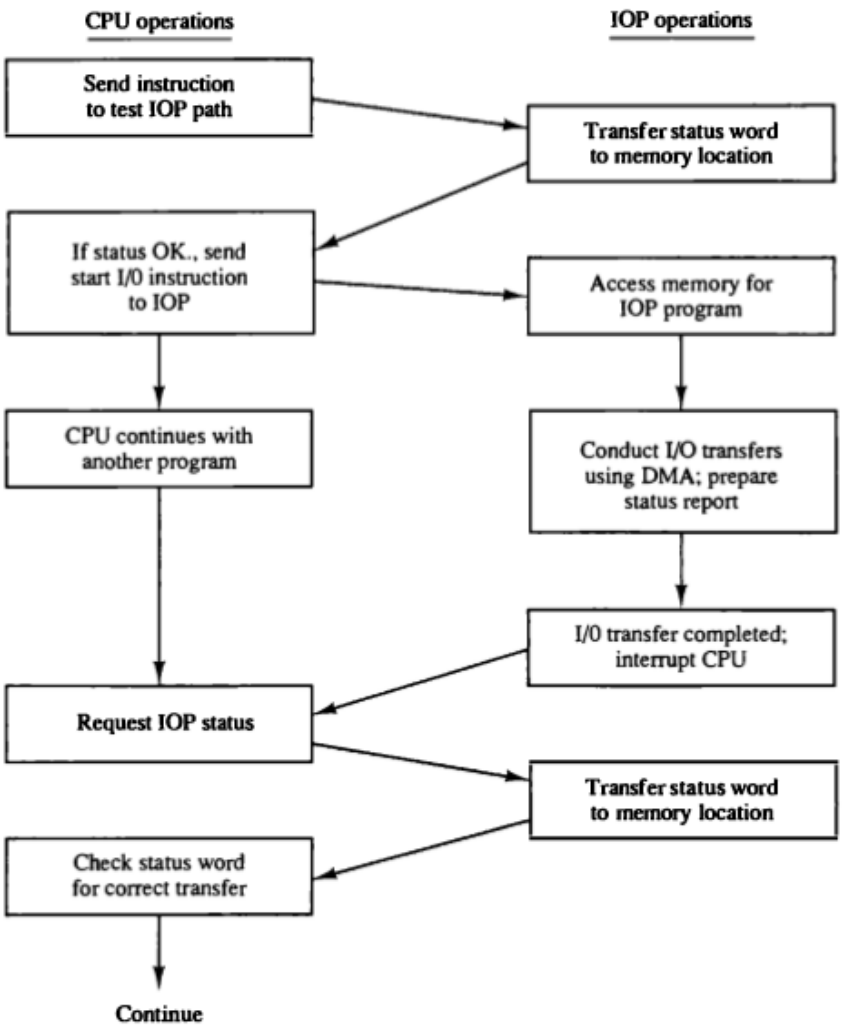


Figure 19: CPU-IOP communication

Unit -V

Assignment-Cum-Tutorial Questions

Section - A

1. Which device can be thought of as transducers which can sense physical effects and convert them into machine-tractable data? [ ]
   a) Storage devices b) Peripheral devices
   c) Both d) None of the above
2. Which devices are usually designed on the complex electromechanical principle? [ ]
   a) Storage devices b) Peripheral devices
   c) Input devices d) All of these
3. The devices with variable speeds are usually connected using asynchronous BUS.
   a) True b) False [ ]
4. The transmission on the asynchronous BUS is also called as _____. [ ]
   a) Switch mode transmission b) Variable transfer
   c) Bulk transfer d) Hand-Shake transmission
5. Asynchronous mode of transmission is suitable for systems with multiple peripheral devices.
   a) True b) False [ ]
6. Which is an important data transfer technique? [ ]
   a) CPU b) DMA c) CAD d) None of these
7. The DMA differs from the interrupt mode by_____. [ ]
   a) The involvement of the processor for the operation
   b) The method accessing the I/O devices
   c) The amount of data transfer possible
   d) Both a and c
8. The DMA transfers are performed by a control circuit called as_____ [ ]
   a) Device interface b) DMA controller
   c) Data controller d) Over looker
9. In DMA transfers, the required signals and addresses are given by _____ [ ]
   a) Processor b) Device drivers
   c) DMA controllers d) The program itself
10. After the completion of the DMA transfer the processor is notified by____ [ ]
    a) Acknowledge signal b) Interrupt signal
    c) WMFC signal d) None of the above
11. _____ register is used for the purpose of controlling the status of each interrupt request in parallel

priority interrupt? [ ]

a) Mass b) Mark c) Make d) Mask

12. Interrupts initiated by an instruction is called as_____ [ ]

a) Internal b) External c) Hardware d) Software

13. The DMA controller has _____ registers? [ ]

a) 4 b) 2 c) 3 d) 1

14. When the R/W bit of the status register of the DMA controller is set to 1? [ ]

a) Read operation is performed

b) Write operation is performed

c) Both Read & Write operations are performed

d) No Read/Write operations are allowed

15. Can a single DMA controller perform operations on two different disks simultaneously…?

a) True b) False c) can't say [ ]

16. When process requests for a DMA transfer, what will happen? [ ]

a) Then the process is temporarily suspended

b) The process continues execution

c) Another process gets executed

d) Both a and c

17. The DMA transfer is initiated by _____ [ ]

a) Processor b) The process being executed

c) I/O devices d) OS

18. In memory-mapped I/O_____? [ ]

a) The I/O devices and the memory share the same address space

b) The I/O devices have a separate address space

c) The memory and I/O devices have an associated address space

d) A part of the memory is specifically set aside for the I/O operation

19. To overcome the lag in the operating speeds of the I/O device and the processor use ___

a) Buffer spaces b) Status flags [ ]

c) Interrupt signals d) Exceptions

20. The method of accessing the I/O devices by repeatedly checking the status flags is____

a) Program-controlled I/O b) Memory-mapped I/O [ ]

c) I/O mapped d) None of the above

21. The method of synchronizing the processor with the I/O device in which the device sends a signal when it is ready is? [ ]

a) Exceptions b) Signal handling

c) Interrupts d) DMA

22. The method which offers higher speeds of I/O transfers is_____ [ ]

a) Interrupts b) Memory mapping

c) Program-controlled I/O d) DMA

23. The return address from the interrupt-service routine is stored in _____ [ ]

a) System heap b) Processor register

c) Processor stack d) Memory

24. The signal sent to the device from the processor to the device after receiving an interrupt is

a) Interrupt-acknowledge b) Return signal [ ]

c) Service signal d) Permission signal

Section - B

1. Explain briefly various peripheral devices used in a system.
2. Explain Asynchronous communication interface with a neat sketch.
3. Explain about programmed I/O for data transfer.
4. Explain block diagram of Input-Output Processor.
5. Explain how CPU and IOP communicate for data transfer?
6. With the help of a neat diagram explain how the priority of a device is decoded parallelly for servicing the interrupts.
7. Define baud Rate.
8. What is vector address?
9. Illustrate parallel transfer of data in asynchronous mode.
10. Explain different types of synchronization mechanisms used in asynchronous communication.
11. With a neat diagram illustrate Daisy chaining priority interrupt.
12. Explain DMA in detail with a neat sketch
13. Differentiate between parallel and serial priority.

Section – C

1. The size of the data count register of a DMA controller is 16 bits. The processor needs to transfer a file of 29,154 kilobytes from disk to main memory. The memory is byte addressable. The minimum number of times the DMA controller needs to get the control of the system bus from the processor to transfer the file from the disk to main memory is _____ **(GATE 2016)**

(A) 3644 (B) 3645 (C) 456 (D) 1823[ ]

2. A hard disk with a transfer rate of 10 Mbytes/ second is constantly transferring data to memory using DMA. The processor runs at 600 MHz, and takes 300 and 900 clock cycles to initiate and complete DMA transfer respectively. If the size of the transfer is 20 Kbytes, what is the percentage of processor time consumed for the transfer operation? **(GATE 2004)**

(A) 5.0% (B) 1.0% (C) 0.5% (D) 0.1%[ ]

UNIT - VI

PARALLEL PROCESSING

Objective:

- To familiarize the concept of pipelining and its efficiency.
- To gain knowledge of multiprocessors and their working.

Syllabus:

Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction Pipeline.

Multi Processors: Characteristics of multiprocessors, interconnection structures, inter processor arbitration, cache coherence

Learning Outcomes:

At the end of the unit student will be able to:

1. Demonstrate the use of pipelining and multiprocessor.

Learning Material

6.1 PARALLEL PROCESSING

- Parallel Processing provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.
- For example, while an instruction is being executed in the ALU, the next instruction can be read from memory. The system may have two or more ALUs and be able to execute two or more instructions at the same time.
- The purpose of parallel processing is to speed up the computer process-ing capability and increase its throughput.
- Parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish between parallel and serial operations by the type of registers used.
- Shift registers operate in serial fashion one bit at a time while registers with parallel load operate with all the bits of the word simultaneously
- Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.
- Parallel processing is established by distributing the data among the multiple functional units. For example, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

- The following figure 1 shows one possible way of separating the execution unit into eight functional units operating in parallel.

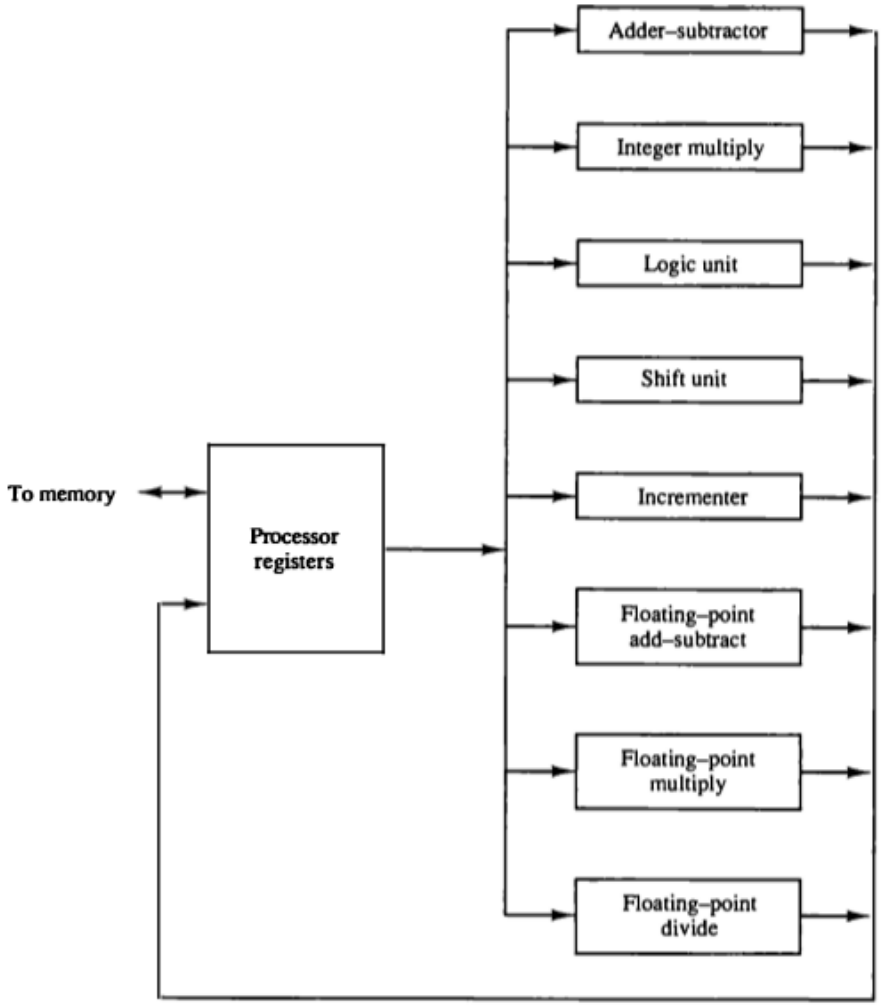


Figure 1: Processor with multiple functional units

- There are a variety of ways that parallel processing can be classified.
- Flynn's classification divides computers into four major groups as follows:
  1. Single instruction stream, single data stream (SISD)
  2. Single instruction stream, multiple data stream (SIMD)
  3. Multiple instruction stream, single data stream (MISD)
  4. Multiple instruction stream, multiple data stream (MIMD)

6.1.1 Single instruction stream, single data stream (SISD)

- SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.

- Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.
- Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

Eg: C = A + B

6.1.2 Single instruction stream, multiple data stream (SIMD)

- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- All processors receive the same instruction from the control unit but operate on different items of data.
- The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

Eg: C[i] = A[i] + B[i]

6.1.3 Multiple instruction stream, single data stream (MISD)

- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

6.1.4 Multiple instruction stream, multiple data stream (MIMD)

- MIMD organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multicomputer systems can be classified in this category.

Parallel processing can be achieved by the following different ways

1. Pipeline processing
2. Vector processing
3. Array processors

6.2 PIPELINING

- Pipelining is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.
- Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- The final result is obtained after the data have passed through all segments.
- The name pipeline implies a flow of information analogous to an industrial assembly line.

- For example, suppose that we want to perform the combined multiply and add operations with a stream of numbers.

  $A_i * B_i + C_i$ for i = 1, 2, 3, ... , 7

- Each sub-operation is to be implemented in a segment within a pipeline.
- Each segment has one or two registers and a combinational circuit as shown in Fig. 2.
- The sub-operations performed in each segment of the pipeline are as follows:

  $R1 \leftarrow A_i$, $R2 \leftarrow B_i$ Input A_i and B_i

  $R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$ Multiply and input C_i

  $R5 \leftarrow R3 + R4$ Add C_i to product

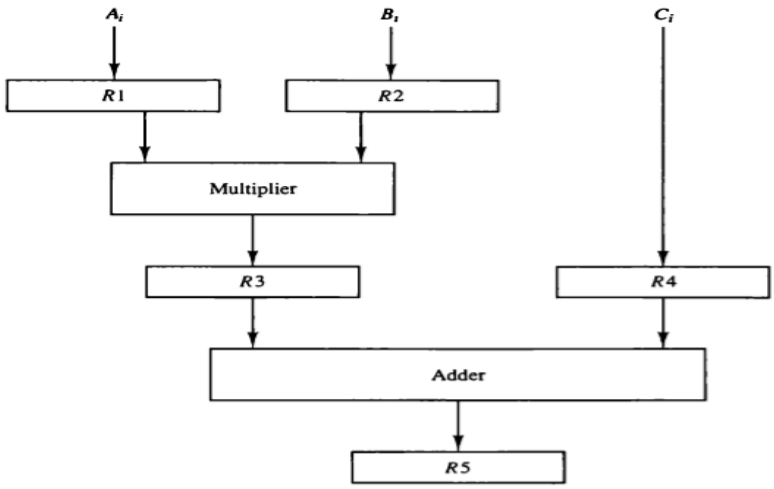


Figure 2: Example of pipeline processing

- The five registers are loaded with new data every clock pulse. The effect of each clock is shown in Table given below.

Table 1: Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

- The first clock pulse transfers A1 and B1 into R1 and R2.
- The second clock pulse transfers the product of R1 and R2 into R3 and C1 into R4. The same clock pulse transfers A2 and B2 into R 1 and R2.
- The third clock pulse operates on all three segments simultaneously. It places A, and B, into R1 and R2, transfers the product of R1 and R2 into R3, transfers C, into R4, and places the sum of R3 and R4 into R5. It takes three clock pulses to fill up the pipe and retrieve the first output from R5.
- From there on, each clock produces a new output and moves the data one step down the pipeline. This happens as long as new input data flow into the system. When no more input data are available, the clock must continue until the last output emerges out of the pipeline.
- The general structure of a four-segment pipeline is illustrated in Fig.3.
- The operands pass through all four segments in a fixed sequence. Each segment consists of a combinational circuit S_i that performs a sub-operation over the data stream flowing through the pipe.
- The segments are separated by registers R_i that hold the intermediate results between the stages.

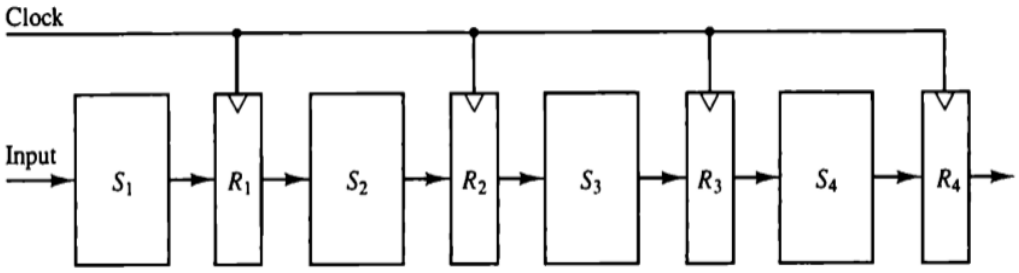


Figure 3: Four segment pipeline

- The behavior of a pipeline can be illustrated with a space-time diagram. This is a diagram that shows the segment utilization as a function of time.
- The space-time diagram of a four-segment pipeline is demonstrated in Fig. 4.

Segment:	1	2	3	4	5	6	7	8	9	→ Clock cycles
1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

Figure 4: Space time diagram for pipeline

- The above figure 4 shows six tasks T1 through T6 executed in four segments.
- Initially, task T1 is handled by segment 1. After the first clock, segment 2 is busy with T1, while segment1 is busy with task T2. Continuing in this manner, the first task T1 is completed after the fourth clock cycle. From then on, the pipe completes a task every clock cycle.

- Consider the case where a k-segment pipeline with a clock cycle time t_p is used to execute n tasks.
- The first task T1 requires a time equal tokt_p to complete its operation since there are *k* segments in the pipe.
- The remaining *n-1* tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to $(n-1)t_p$.
- Therefore, to complete *n* tasks using a k-segment pipeline requires $k + (n-1)$ clock cycles.
- For example, the diagram shows four segments and six tasks. The time required to complete all the operations is 4 + (6 - 1) = 9 clock cycles, as indicated in the diagram.
- Consider a non pipeline unit that performs the same operation and takes a time equal to tn. to complete each task. The total time required for n tasks is nt_n. The speedup of a pipeline processing over an equivalent non pipeline processing is defined by the ratio:

$$S = \frac{nt_n}{(k + n - 1)t_p}$$

- As the number of tasks increases, *n* becomes much larger than $k - 1$, and $k + n - 1$ approaches the value of *n*. Under this condition, the speedup becomes

$$S = \frac{n\,t_n}{n\,t_p} \quad \Rightarrow \quad S = \frac{t_n}{t_p}$$

- If we assume that the time it takes to process a task is the same in the pipeline and non pipeline circuits, we will have $t_n = kt_p$. Including this assumption, the speedup reduces to:

$$S = \frac{kt_p}{t_p} = k$$

- This shows that the theoretical maximum speed up that a pipeline can provide is k, where k is the number of segments in the pipeline.
- To duplicate the theoretical speed advantage of a pipeline process by means of multiple functional units, it is necessary to construct k identical units that will be operating in parallel.
- In the following Fig. 5, where four identical circuits are connected in parallel.

- Each P circuit performs the same task of an equivalent pipeline circuit.
- Instead of operating with the input data in sequence as in a pipeline, the parallel circuits accept four input data items simultaneously and perform four tasks at the same time

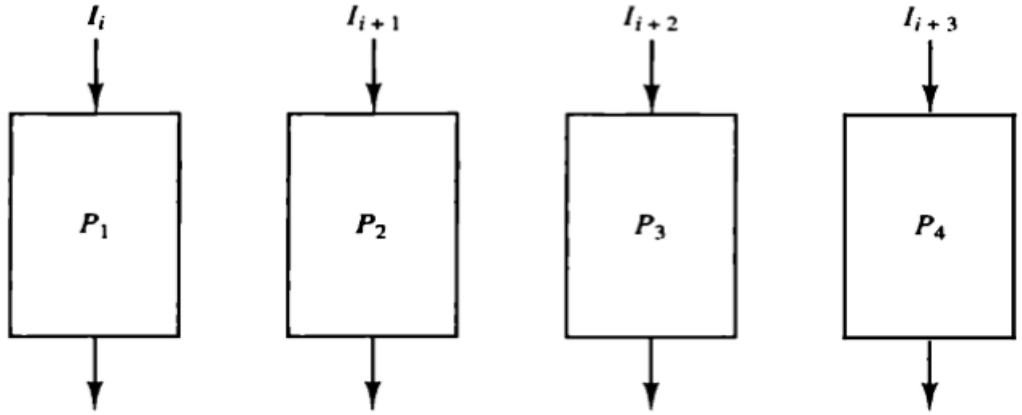


Figure 5: Multiple functional units in parallel

- In the above figure 5, four identical circuits are connected in parallel.
- The implication is that a k-segment pipeline processor can be expected to equal the performance of k copies of an equivalent non pipeline circuit under equal operating conditions.
- Each P circuit performs the same task of an equivalent pipeline circuit.
- Instead of operating with the input data in sequence as in a pipeline, the parallel circuits accept four input data items simultaneously and perform four tasks at the same time.

There are two areas of computer design where the pipeline organization is applicable.

1. Arithmetic pipeline
2. Instruction pipeline

6.2.1. Arithmetic pipeline

- Pipeline arithmetic units are usually found in very high speed computers.
- They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.
- The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

$$X = A \times 2^a$$
$$Y = B \times 2^b$$

- The floating-point addition and subtraction can be performed in four segments, as shown in Figure 6.

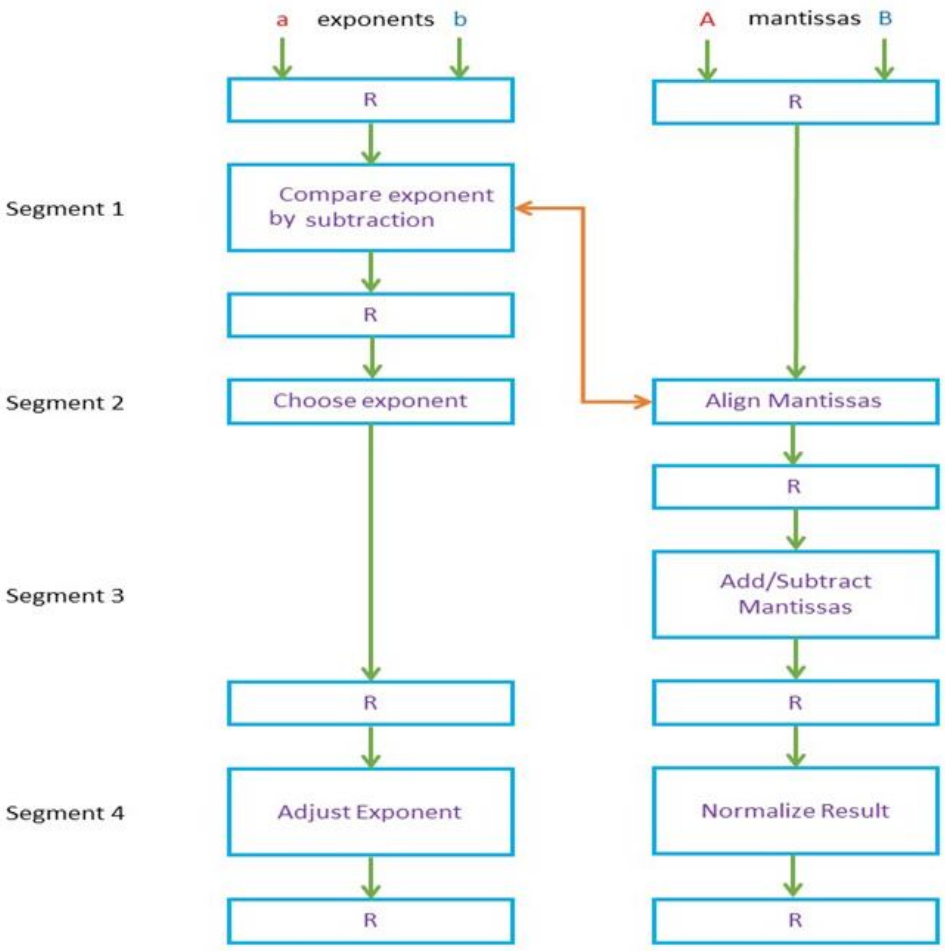


Figure 6: Pipeline for floating point addition and subtraction operation

- The registers labeled R are placed between the segments to store intermediate results.
- The sub-operations that are performed in the four segments are:
    1. Compare the exponents.
    2. Align the mantissas.
    3. Add or subtract the mantissas.
    4. Normalize the result.
- The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.
- The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right. This produces an alignment of the two mantissas.
- The two mantissas are added or subtracted in segment 3.

- The result is normalized in segment 4. When an overflow occurs, the mantissa of the sum or difference is shifted right and the exponent incremented by one. If an underflow occurs, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

6.2.2. Instruction pipeline

- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- The computer needs to process each instruction with the following sequence of steps.
  1. Fetch the instruction from memory.
  2. Decode the instruction.
  3. Calculate the effective address.
  4. Fetch the operands from memory.
  5. Execute the instruction.
  6. Store the result in the proper place.
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration.
- The time that each step takes to fulfill its function depends on the instruction and the way it is executed.
- Example: Four-Segment Instruction Pipeline.
- Figure 7 shows how the instruction cycle in the CPU can be processed with a four-segment pipeline.

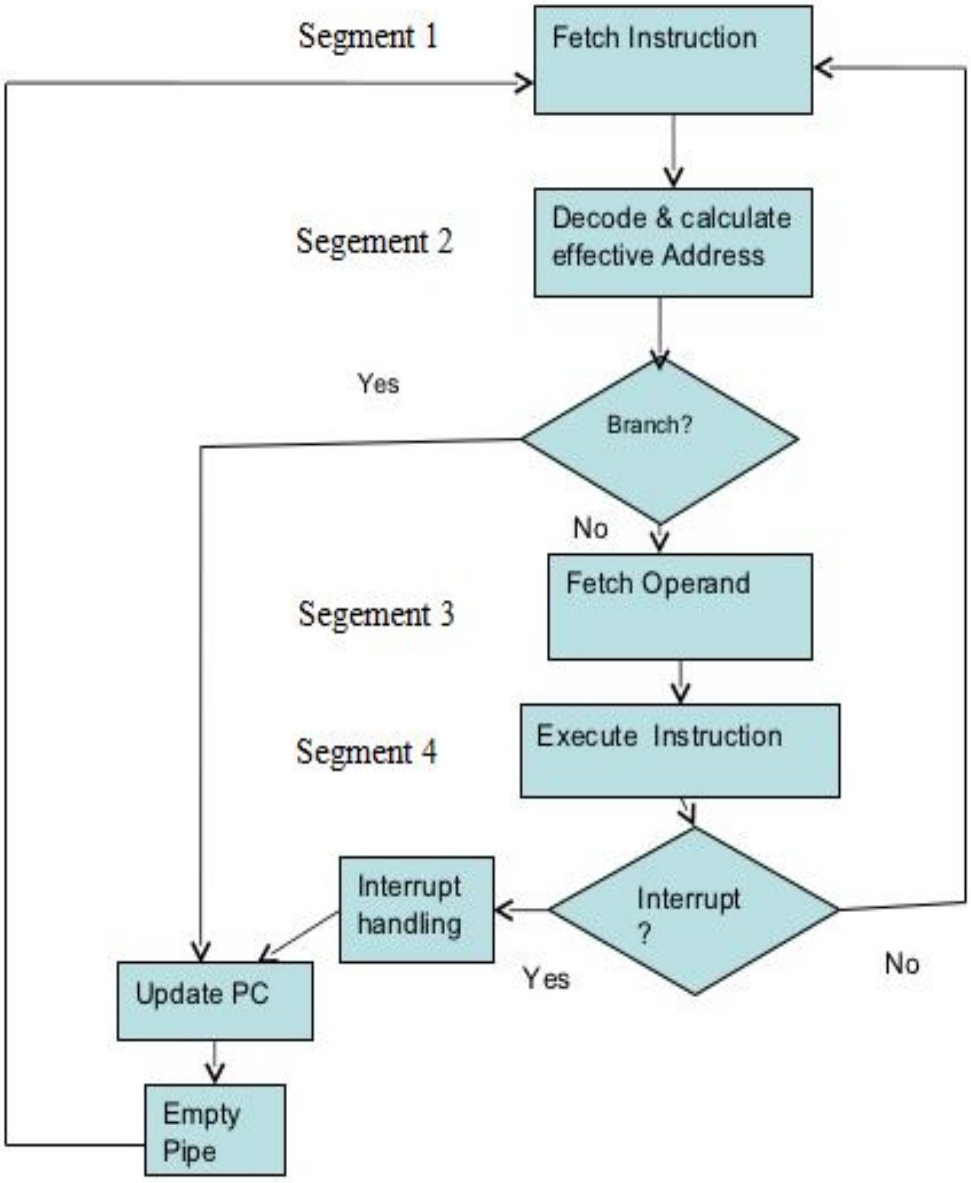


Figure 7: Four segment CPU Pipeline

- The following figure 8 shows the operation of the instruction 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 8: Timing of instruction pipeline

- The time in the horizontal axis is divided into steps of equal duration.
- The four segments are represented in the diagram with an abbreviated symbol.
  1. FI is the segment that fetches an instruction.
  2. DA is the segment that decodes the instruction and calculates the effective address.
  3. FO is the segment that fetches the operand.
  4. EX is the segment that executes the instruction.
- It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.
- In the absence of a branch instruction, each segment operates on different instructions. Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FL.
- Assume now that instruction 3 is a branch instruction. As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step 6. If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken, the instruction fetched previously in step 4 can be used. The pipeline then continues until a new branch instruction is encountered.
- A delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand.
- In that case, segment FO must wait until segment EX has finished its operation.

6.3 CHARACTERISTICS OF MULTIPROCESSORS

- A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment.
- The term "processor" In multiprocessor can mean either a central processing unit (CPU) or an input-output processor (lOP).
- However, a system with a single CPU and one or more lOPs is usually not included in the definition of a multiprocessor system unless the lOP has computational facilities comparable to a CPU.
- As it is most commonly defined, a multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more lOPs.
- Multiprocessors are classified as Multiple Instruction stream, Multiple Data stream (MIMD) systems.
- There are some similarities between multiprocessor and multicomputer systems since both support concurrent operations.
- However, there exists an important distinction between a system with multiple computers and a system with multiple processors.
- Computers are interconnected with each other by means of communication lines to form a computer network.
- A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.
- Multiprocessing improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system.
- If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled processor.
- Multiprocessors are classified by the way their memory is organized as follows.
  1. Tightly coupled
  2. loosely coupled

6.3.1 Tightly coupled

- A multiprocessor system with common shared memory is classified as a shared memory or tightly coupled multiprocessor.

6.3.2 Loosely coupled

- An alternative model of microprocessor is the distributed-memory or loosely coupled system.
- Here each processor element in a loosely coupled system has its own private local memory.

6.4 INTERCONNECTION STRUCTURES

- There are several physical forms available for establishing an interconnection network. Some of these schemes follows:
  1. Time-shared common bus
  2. Multiport memory
  3. Crossbar switch

6.4.1. Time-shared common bus

- A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit.
- Only one processor can communicate with the memory or another processor at any given time.

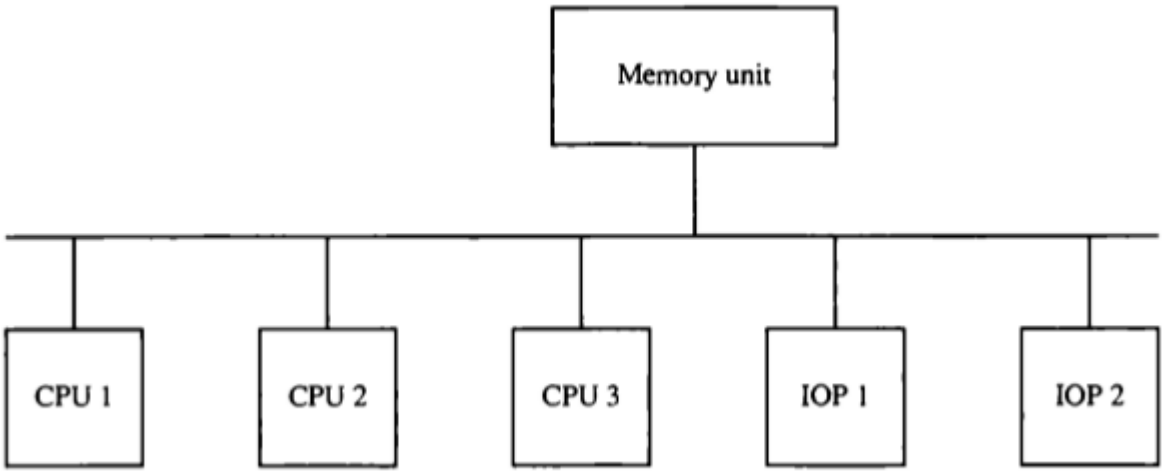


Figure 9: Time shared common bus organization

- Any processor wishing to initiate a transfer must first determine the availability status of the bus, and only after the bus becomes available can the processor address the destination unit to initiate the transfer.
- A command is issued to inform the destination unit what operation is to be performed. The receiving unit recognizes its address in the bus and responds to the control signals from the sender, after which the transfer is initiated.
- The system may exhibit transfer conflicts since one common bus is shared by all processors. These conflicts must be resolved by incorporating a bus controller that establishes priorities among the requesting units.
- A single common-bus system is restricted to one transfer at a time.
- The total overall transfer rate within the system is limited by the speed of the single path.

- Dual bus structure is depicted in Fig.10. Here we have a number of local buses each connected to its own local memory and to one or more processors.
- Each local bus may be connected to a CPU, anlOP, or any combination of processors.
- A system bus controller links each local bus to a common system bus.
- The I/O devices connected to the local lOP, as well as the local memory, are available to the local processor.

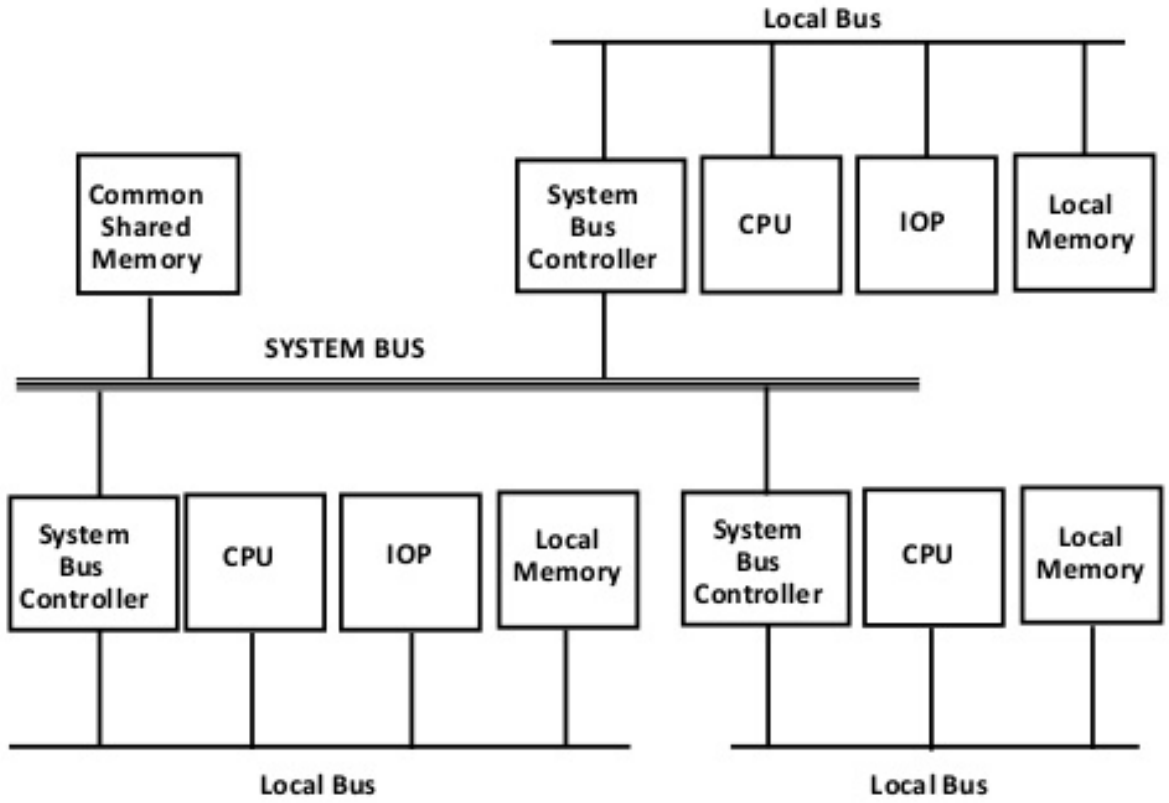


Figure 10: System bus structure for multiprocessors

- The memory connected to the common system bus is shared by all processors.
- If an lOP is connected directly to the system bus, the I/O devices attached to it may be made available to all processors.
- Only one processor can communicate with the shared memory and other common resources through the system bus at any given time.
- In this way, the average access time of the local memory can be made to approach the cycle time of the CPU to which it is attached.

6.4.2. Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU.
- This is shown in Fig 11, for four CPUs and four memory modules (MMs).
- Each processor bus is connected to each memory module.
- A processor bus consists of the address, data, and control lines required to communicate with memory.
- The memory module is said to have four ports and each port accommodates one of the buses.

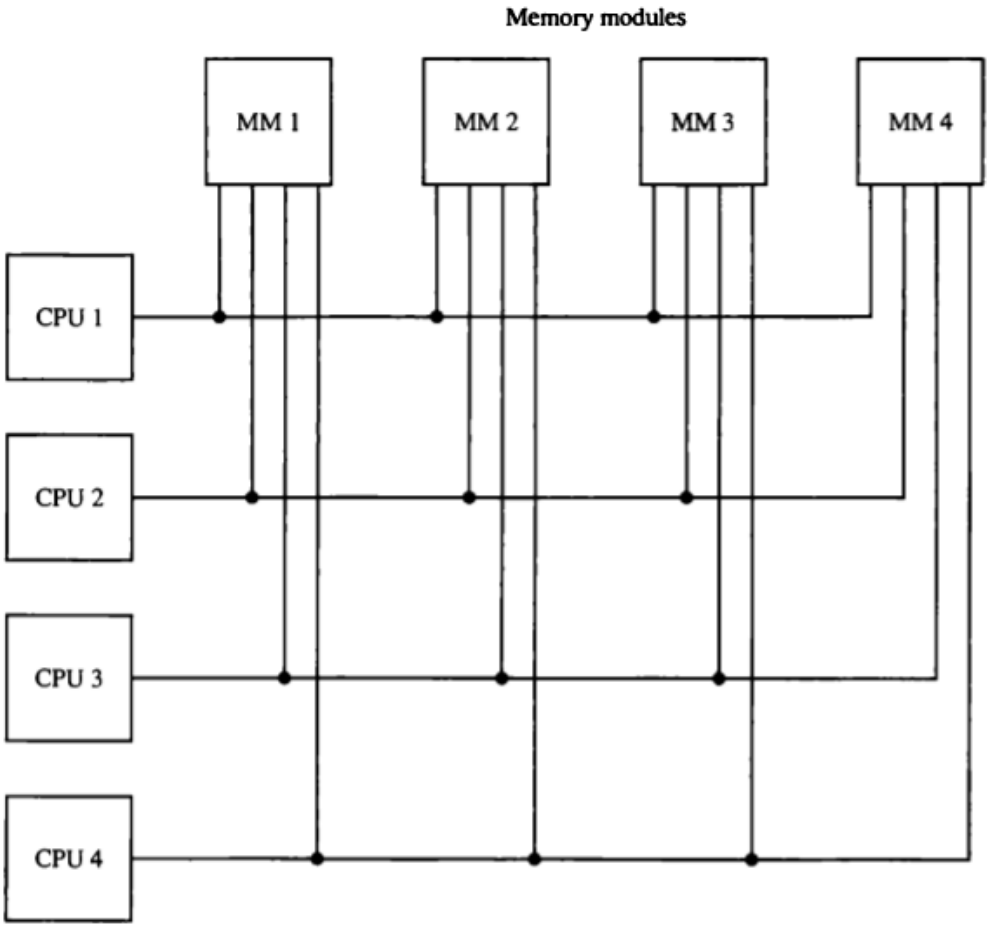


Figure 11: Multiport memory organization

- Memory access conflicts are resolved by assigning fixed priorities to each memory port.
- The priority for memory access associated with each processor may be established by the physical port position that its bus occupies in each module.
- Thus CPU 1 will have priority over CPU 2, CPU 2 will have priority over CPU 3, and CPU 4 will have the lowest priority.

6.4.3. Crossbar Switch

- The crossbar switch organization consists of a number of cross-points that are placed at intersections between processor buses and memory module paths.
- Figure 12 shows a crossbar switch interconnection between four CPUs and four memory modules.
- The small square in each cross-point is a switch that determines the path from a processor to a memory module.

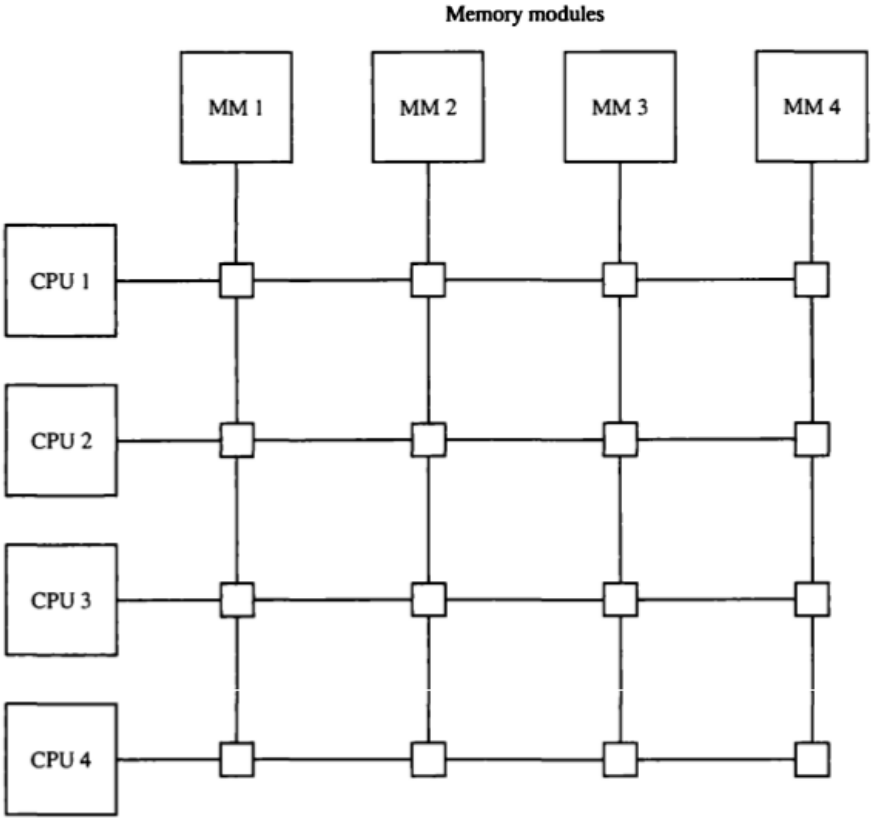


Figure 12: Crossbar switch

- Each switch point has control logic to set up the transfer path between a processor and memory.
- Switch point examines the address that is placed in the bus to determine whether its particular module is being addressed. It also resolves multiple requests for access to the same memory module on a predetermined priority basis.
- The following figure 13 shows the functional design of a crossbar switch connected to one memory module. The circuit consists of multiplexers that select the data, address, and control from one CPU for communication with the memory module.

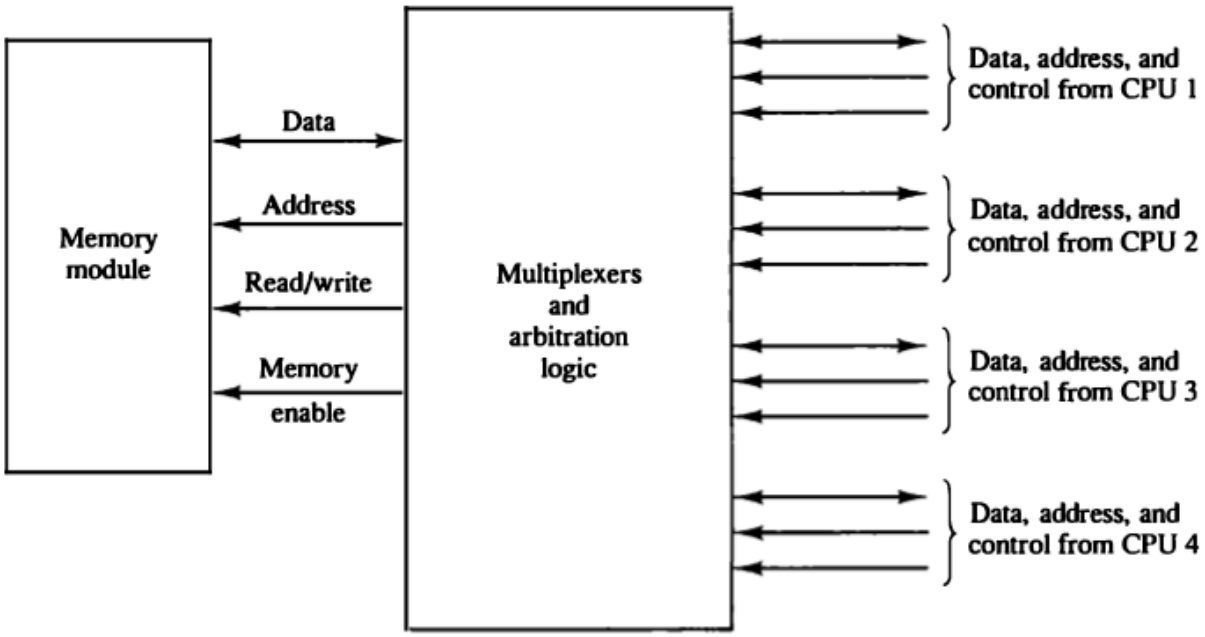


Figure 13: Block diagram of crossbar switch

- Priority levels are established by the arbitration logic to select one CPU when two or more CPUs attempt to access the same memory.
- A crossbar switch organization supports simultaneous transfers from memory modules because there is a separate path associated with each module.

6.5 INTERPROCESSOR ARBITRATION

- A bus that connects major components in a multiprocessor system, such as CPUs, lOPs, and memory, is called a system bus.
- The processors in a shared memory multiprocessor system request access to common memory or other common resources through the system bus.
- If no other processor is currently utilizing the bus, the requesting processor may be granted access immediately.
- However, the requesting processor must wait if another processor is currently utilizing the system bus. Furthermore, other processors may request the system bus at the same time.
- Arbitration must then be performed to resolve this multiple contention for the shared resources.
- Arbitration must then be performed to resolve this multiple contention for the shared resources.
- Arbitration procedures service all processor requests on the basis of established priorities.
- A hardware bus priority resolving technique can be established by means of following ways.

1. Static Arbitration
    i. Serial Arbitration Procedure
    ii. Parallel Arbitration Logic
2. Dynamic Arbitration

6.5.1.1 Serial Arbitration Procedure

- The serial priority resolving technique is obtained from a daisy-chain connection of bus arbitration circuits similar to the priority interrupt logic.
- The processors connected to the system bus are assigned priority according to their position along the priority control line.
- When multiple devices concurrently request the use of the bus, the device with the highest priority is granted access to it.

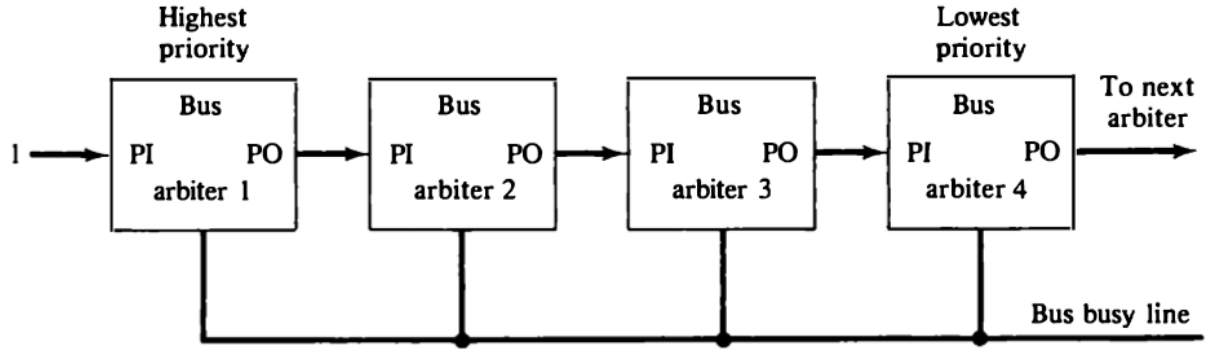


Figure 14: Serial (Daisy-chain) arbitration

- A processor may be in the middle of a bus operation when a higher priority processor requests the bus. The lower-priority processor must complete its bus operation before it hand over control of the bus.
- The bus busy line shown in above figure 14 provides a mechanism for an orderly transfer of control.

6.5.1.2 Parallel Arbitration Logic

- The parallel bus arbitration technique uses an external priority encoder and a decoder as shown in following figure 15.
- Each bus arbiter in the parallel scheme has a bus request output line and a bus acknowledge input line.
- Each arbiter enables the request line when its processor is requesting access to the system bus.
- The processor takes control of the bus if its acknowledge input line is enabled. The bus busy line provides an orderly transfer of control, as in the daisy-chaining case.

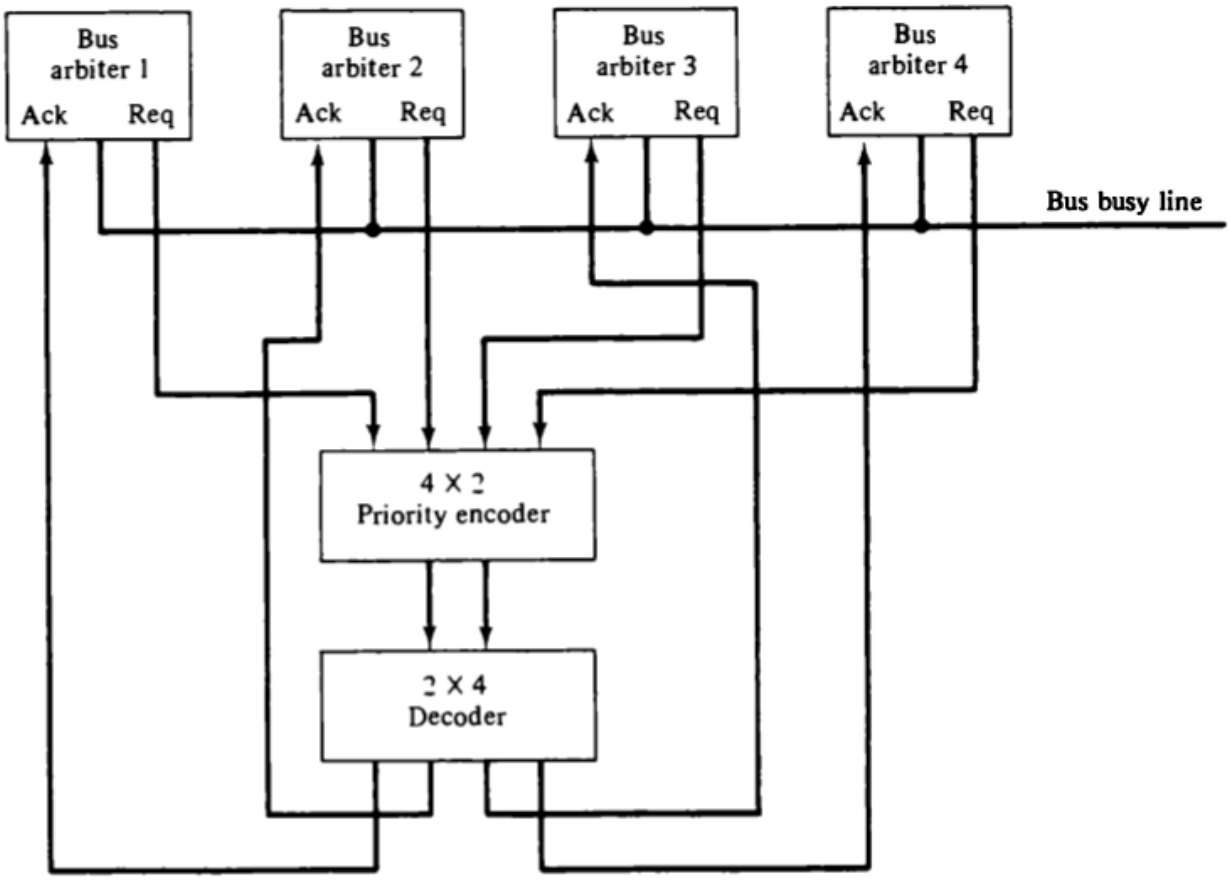


Figure 15: Parallel arbitration

- The request lines from four arbiters going into a 4x2 priority encoder.
- The output of the encoder generates a 2-bit code which represents the highest-priority unit among those requesting the bus.
- The 2-bit code from the encoder output drives a 2x4 decoder which enables the proper acknowledge line to grant bus access to the highest-priority unit.

6.5.2 Dynamic Arbitration

- The two bus arbitration procedures just described use a static priority algorithm since the priority of each device is fixed by the way it is connected to the bus.
- Dynamic priority algorithm gives the system the capability for changing the priority of the devices while the system is in operation.
- A few arbitration procedures that use dynamic priority algorithms:

  1. Time slice
  2. Polling
  3. LRU
  4. FIFO

5. Rotating daisy-chain

6.5.2.1 Time slice

- The time slice algorithm allocates a fixed-length time slice of bus time that is offered sequentially to each processor, in round-robin fashion.

6.5.2.2 Polling

- In polling, the bus grant signal is replaced by a set of lines called poll lines which are connected to all units.
- These lines are used by the bus controller to define an address for each device connected to the bus.
- When a processor that requires access recognizes its address, it activates the bus busy line and then accesses the bus

6.5.2.3 LRU

- The least recently used (LRU) algorithm gives the highest priority to the requesting device that has not used the bus for the longest interval.

6.5.2.4 FIFO

- In the first-come, first-serve (FIFO) scheme, requests are served in the order received

6.5.2.5 Rotating daisy-chain

- The rotating daisy-chain procedure is a dynamic extension of the daisy-chain algorithm.
- In this scheme there is no central bus controller, and the priority line is connected from the priority-out of the last device back to the priority-in of the first device in a closed loop.

6.6 CACHE COHERENCE

- If the operation is to write, there are two commonly used procedures to update memory.
- In the *write-through* policy, both cache and main memory are updated with every write operation.
- In the *write-back* policy, only the cache is updated and the location is marked so that it can be copied later into main memory.
- In a shared memory multiprocessor system, all the processors share a common memory. In addition, each processor may have a local memory, part or all of which may be a cache.
- The same information may reside in a number of copies in some caches and main memory. To ensure the ability of the system to execute memory operations correctly, the multiple copies must be kept identical. This requirement imposes a cache *coherence problem*.
- A memory scheme is coherent if the value returned on a load instruction is always the value given by the latest store instruction with the same address.

6.6.1 Conditions for Incoherence

- Cache coherence problems exist in multiprocessors with private caches because of the need to share writable data.
- Consider the three-processor configuration with private caches shown in figure 16.
- Assume that X contains the value of 52. The load on X to the three processors results in consistent copies in the caches and main memory.

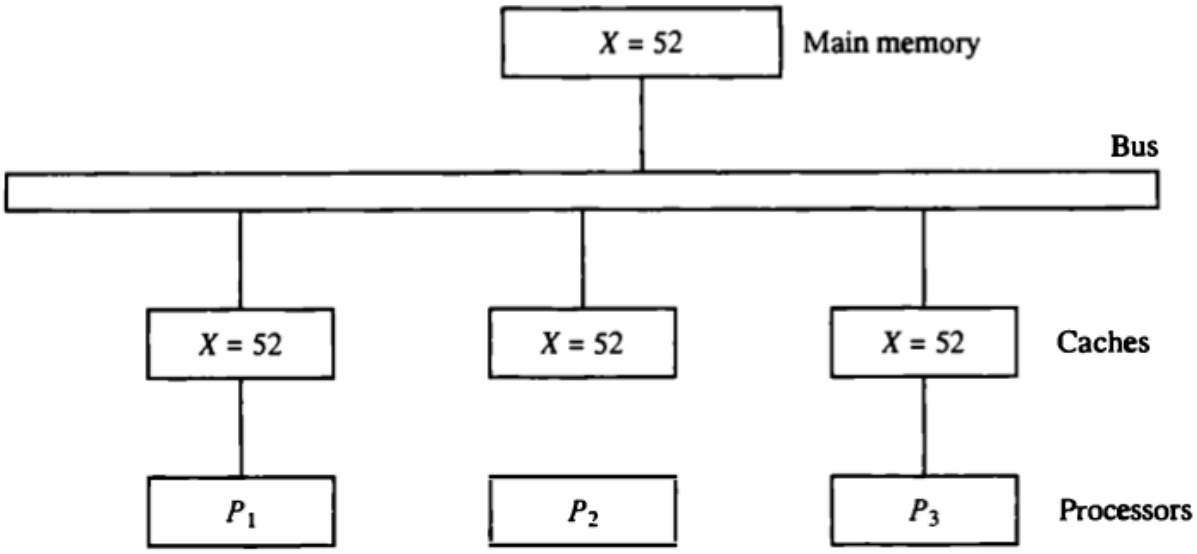


Figure 16: Cache configuration after a load on X

- If one of the processors performs a store to X, the copies of X in the caches become inconsistent.
- A load by the other processors will not return the latest value.
- Depending on the memory update policy used in the cache, the main memory may also be inconsistent with respect to the cache as shown in the following figure 17.

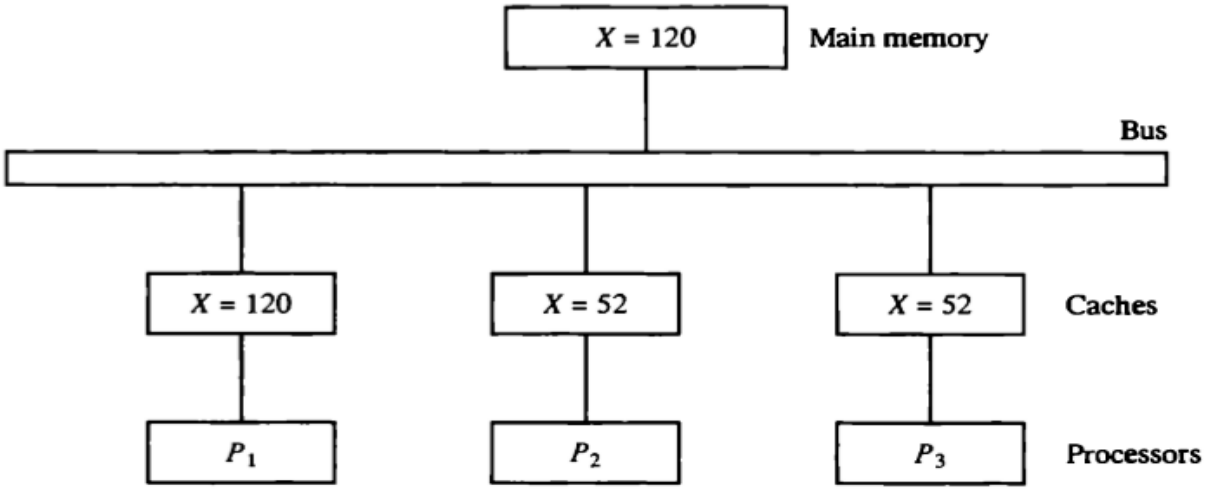


Figure 17: Cache configuration after a store to X by processor P1 With write-through cache policy

- A store to X (of the value of 120) into the cache of processor P1 updates memory to the new value in a write-through policy.
- A write-through policy maintains consistency between main memory and the originating cache, but the other two caches are inconsistent since they still hold the old value.
- In a write-back policy, main memory is not updated at the time of the store. The copies in the other two caches and main memory are inconsistent.

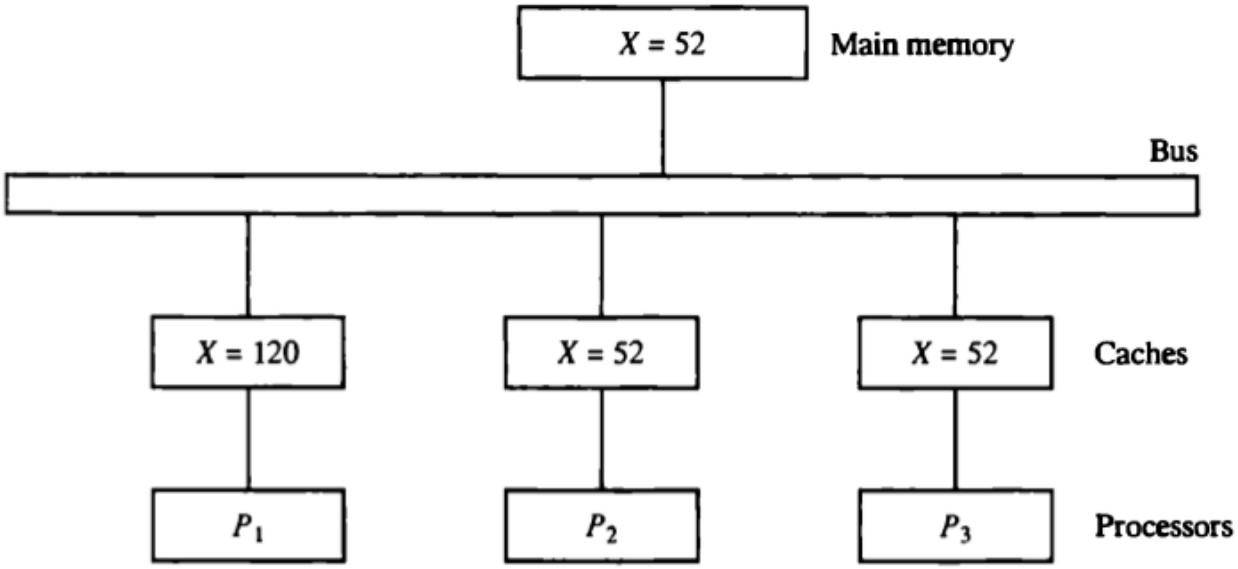


Figure 18: Cache configuration after a store to X by processor P1 With write-back cache policy

- Another configuration that may cause consistency problems is a direct memory access (DMA) activity in conjunction with an lOP connected to the system bus.
- In the case of input, the DMA may modify locations in main memory that also reside in cache without updating the cache.
- During a DMA output, memory locations may be read before they are updated from the cache when using a write-back policy.

6.6.2 Solutions to the Cache Coherence Problem

1. A simple scheme is to disallow private caches for each processor and have a shared cache memory associated with main memory. For performance considerations it is required to attach a private cache to each processor.
2. One scheme that has been used allows only non-shared and read-only data to be stored in caches. Such items are called cachable. Shared writable data are non-cachable. The compiler must tag data as either cachable or noncachable, and the system hardware makes sure that only cachable data are stored in caches. The non-cachable data remain in main memory.

3. A scheme that allows writable data to exist in at least one cache is a method that employs a centralized global table in its compiler. The status of memory blocks is stored in the central global table. Each block is identified as Read-Only (RO) or Read and write (RW). All caches can have copies of blocks identified as RO. Only one cache can have a copy of an RW block. Thus if the data are updated in the cache with an RW block, the other caches are not affect because they do not have a copy of this block.

   The cache coherence problem can be solved by means of a combination of software and hardware or by means of hardware-only schemes. The two methods mentioned previously use software based procedures.

4. In the hardware solution, the cache controller is specially designed to allow it to monitor all bus requests from CPUs and lOPs. All caches attached to the bus constantly monitor the network for possible write operations. The bus controller that monitors this action is referred to as a snoopy cache controller. This is basically a hardware unit designed to maintain a bus-watching mechanism over all the caches attached to the bus. Various schemes have been proposed to solve the cache coherence prob-lem by means of snoopy cache protocol.
   - When a word in a cache is updated by writing into it, the corresponding location in main memory is also updated. The local snoopy controllers in all other caches check their memory to determine if they have a copy of the word that has been overwritten.
   - If a copy exists in a remote cache, that location is marked invalid. Because all caches snoop on all bus writes, whenever a word is written, the net effect is to update it in the original cache and main memory and remove it from all other caches.
   - If at some future time a processor accesses the invalid Item from its cache, the response is equivalent to a cache miss, and the updated Item is transferred from main memory. In this way, inconsistent versions are prevented.

Unit - VI

Assignment-Cum-Tutorial Questions

Section - A

1. When multiple-instructions are overlapped during execution of program, then function performed is called_____? [ ]

   a) Multitasking b) Multiprogramming

   c) Hardwired control d) Pipelining

2. Pipelining increases CPU instruction_____? [ ]

   a) Size b) Through put c) Cycle rate d) Time

3. Each stage in pipelining should be completed within ____ cycle. [ ]

   a) 1 b) 2 c) 3 d) 4

4. The pipelining process is also called as _____. [ ]

   a) Superscalar operation b) Assembly line operation

   c) Vonneumann cycle d) None of the mentioned

5. In pipelining the task which requires the least time is performed first. [ ]

   a) True b) False

6. If a unit completes its task before the allotted time period, then_____. [ ]

   a) It'll perform some other task in the remaining time

   b) Its time gets reallocated to different task

   c) It'll remain idle for the remaining time

   d) None of the mentioned

7. An instruction that does no operation for changing state is known as_____? [ ]

   a) Nope b) No c) No-op d) Nop

8. Which of the following is not a Flynn's classification? [ ]

   a) Single instruction stream, single data stream

b) Single instruction stream, multiple data stream

c) Multiple Input, Multiple Output stream

d) Multiple instruction stream, multiple data stream

9. Which of the following is not a way to achieve parallel processing? [ ]

a) Pipeline processing b) Vector processing

c) Array processors d) Link processing

10. Define parallel processing.

11. Define critical section.

12. Define tightly coupled multiprocessors.

Section - B

1. Explain in detail about three segment instruction pipeline.
2. Describe various ways of handling branch hazards in instruction pipelining.
3. Explain how the device is prioritized serially for granting the bus.
4. Explain the characteristics of multiprocessors.
5. Classify the ways for organizing memory in multiprocessors.
6. Explain Flynn's classification of systems.
7. Draw the diagram for four segment pipeline with clock.
8. Implement floating point addition and subtraction algorithm using three segment pipeline.
9. Draw and explain the functioning of a processor with multiple functional units.
10. Illustrate how to connect 4 processors and 4 memory modules using a crossbar switch.
11. Give reasons for cache inconsistency. How to overcome it?

Section -C

1. The performance of a pipelined processor suffers if _____. **(GATE 2002)** [ ]

a) The pipeline stages have different delays

b) Consecutive instructions are dependent on each other

c) The pipeline stages share hardware resources

d) All of the above

2. Comparing the time T1 taken for a single instruction on a pipelined CPU with time T2 taken on a non-pipelined but identical CPU, we can say that_____ **(GATE 2000)** [ ]

a) T1 <= T2 b) T1 >= T2

c) T1 < T2 d) T1 is T2 plus the time taken for one instruction fetch cycle

1. A 4-stage pipeline has the stage delays as 150, 120, 160 and 140 nanoseconds respectively. Registers that are used between the stages have a delay of 5 nanoseconds each. Assuming constant clocking rate, the total time taken to process 1000 data items on this pipeline will be_____ **(GATE 2004)**

a) 120.4 microseconds b) 160.5 microseconds [ ]

c) 165.5 microseconds d) 590.0 microseconds

4. A 5 stage pipelined CPU has the following sequence of stages:

IF — Instruction fetch from instruction memory,

RD — Instruction decode and register read,

EX — Execute: ALU operation for data and address computation,

MA — Data memory access - for write access, the register read at RD stage is used,

WB — Register write back.

Consider the following sequence of instructions:

I1: LD R0, 1oc1; R0 ← M[1oc1]

I2: ADD R0, R0; R0 ← R0 + R0

I3: SUB R2, R0; R2 ← R2 - R0

Let each stage take one clock cycle.

What is the number of clock cycles taken to complete the above sequence of instructions starting from the fetch of *I1*? **(GATE 20005)**

[ ]

a) 8 b) 10 c) 12 d) 15

5. Consider a 6-stage instruction pipeline, where all stages are perfectly balanced. Assume that there is no cycle-time overhead of pipelining. When an application is executing on this 6-stage pipeline, the speedup achieved with respect to non-pipelined execution if 25% of the instructions incur 2 pipeline stall cycles is_____? **(GATE 2014)** **[ ]**

a) 4 b) 8 c) 6 d) 7