Hall Ticket Number:

II/IV B.Tech (Regular\Supplementary) DEGREE EXAMINATION

February, 2023 **Common to CSE/CB/DS & IT Branches**

Third Semester **Computer Organization**

Time: 3 Hours Maximum Marks:70

Answer question 1 compulsory. **(14X1 = 14 Marks)**

Answer one question from each unit. **(4X14=56 Marks)**

1.	a)	Convert $(F3)_{16}$ into decimal.	CO1	L2	1M
	b)	State the formulas for (r-1)'s Complement and r's Complement	CO1	L1	1M
	c)	What is register transfer language?	CO1	L1	1M
	d)	Name any four logic microoperations.	CO1	L1	1M
	e)	Define instruction code and operation code.	CO2	L1	1M
	f)	List out the memory-reference instructions.	CO2	L1	1M
	g)	How to represent control variables?	CO2	L2	1M
	h)	Show the microinstruction format for the control memory.	CO2	L1	1M
	i)	State the operations on a stack.	CO3	L1	1M
	j)	What are the most common fields found in instruction format?	CO3	L1	1M
	k)	Expand RISC and CISC.	CO3	L3	1M
	l)	When is status command used?	CO4	L2	1M
	m)	Define bootstrap loader.	CO4	L1	1M
	n)	Show the connection of I/O bus to input-output devices.	CO4	L3	1M
		Unit –I			
2.	a)	Draw the arithmetic logic shift unit and show the function table for arithmetic logic shift unit.	CO1	L1	7M
	b)	What are the number systems conversions available? Explain with an example.	CO1	L2	7M
		(OR)			
3.	a)	What are the different ways to implement a common bus system and explain with a neat sketch	CO1	L2	7M
	b)	Label the diagram for 4-bit binary adder and 4-bit adder-subtracter.	CO1	L3	7M
		Unit –II			
4.	a)	Name the registers for the basic computer with number of bits used and describe their functionality.	CO2	L3	7M
	b)	Interpret the symbols and binary code used for microinstruction fields.	CO2	L2	7M
		(OR)			
5.	a)	State the phases of an instruction cycle? Design the flowchart for instruction cycle	CO2	L1	7M
	b)	Show the block diagram of the microprogram sequencer and discuss.	CO2	L2	7M
		Unit –III			
6.	a)	Examine the procedure involved in reverse polish notation with an example.	CO3	L3	7M
	b)	Inspect the hardware for signed-magnitude addition and subtraction.	CO3	L2	7M
		(OR)			
7.	a)	List out any seven addressing modes and interpret each addressing mode with syntax.	CO3	L2	7M
	b)	Display the flowchart for Booth multiplication operation and discuss the operations performed.	CO3	L4	7M
		Unit –IV			
8.	a)	Examine the working of associate memory with a neat diagram.	CO4	L2	7M
	b)	Illustrate the mapping procedures while considering the organization of cache memory.	CO4	L3	7M
		(OR)			
9.		Analyse the various modes of data transfer to and from peripherals	CO4	L2	14M

SCHEME

II/IV B.Tech (Regular\Supplementary) DEGREE EXAMINATION

February, 2023 — **Common to CSE/CB/DS & IT Branches**

Third Semester — **Computer Organization**

Time: 3 Hours — Maximum Marks:70

Answer question 1 compulsory. **(14X1 = 14 Marks)**

Answer one question from each unit. **(4X14=56 Marks)**

1. a) Convert $(F3)_{16}$ into decimal. **CO1 L2 1M**

$(F3)_{16} = (243)_{10}$

b) State the formulas for (r-1)'s Complement and r's Complement **CO1 L1 1M**

r's Complement of N = r^n-N

(r-1)'s Complement of N = (r^n-1)-N

N : given number
r : base
n : digit number

c) What is register transfer language? **CO1 L1 1M**

The Register Transfer Language is the symbolic representation of notations used to specify the Sequence of micro-operations.

d) Name any four logic microoperations. **CO1 L1 1M**

AND (∧), OR (∨), XOR (⊕), Complement/NOT.

e) Define instruction code and operation code. **CO2 L1 1M**

Instruction code is a group of bits that tells the computer to perform a specific operation part.
The operation code of an instruction is a group of bits that define operations such as add, subtract, Multiply, shift and compliment.

f) List out the memory-reference instructions. **CO2 L1 1M**

AND, ADD, LDA, STA, BUN, BSA, ISZ

g) How to represent control variables? **CO2 L2 1M**

Binary variables specify micro operations

h) Show the microinstruction format for the control memory. CO2 L1 1M

F1	F2	F3	CD	BR	AD
3bits	3 bits	3 bits	2 bits	2 bits	7 bits

F1, F2, F3: Micro operation fields

CD: Condition for branching

BR: Branch field

AD: Address field

i) State the operations on a stack. CO3 L1 1M

Push & Pop

j) What are the most common fields found in instruction format? CO3 L1 1M

The most common fields are:-

Operation field: - specifies the operation to be performed like addition.
Address field: - which contains the location of the operand, i.e., register or memory location.
Mode field: - which specifies how operand is to be founded.

k) Expand RISC and CISC. CO3 L3 1M

RISC (reduced instruction set computer) and CISC (complex instruction set computer).

l) When is status command used? CO4 L2 1M

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.

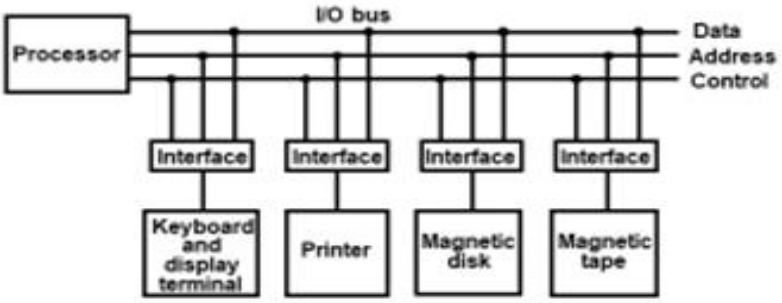
During the transfer, one or more errors may occur which are detected by the interface.

These errors are designated by setting bits in a status register that the processor can read at certain intervals

m) Define bootstrap loader. CO4 L1 1M

Bootstrap loader is a program that resides in the computer's EPROM, ROM, or another non-volatile memory. It is automatically executed by the processor when turning on the computer. The bootstrap loader reads the hard drives boot sector to continue to load the computer's operating system.

n) Show the connection of I/O bus to input-output devices. **CO4 L3 1M**



Unit –I

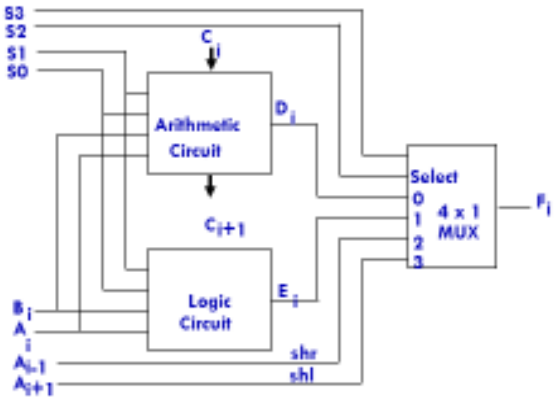
2. a) Draw the arithmetic logic shift unit and show the function table for arithmetic logic shift unit. **CO1 L1 7M**

ARITHMETIC LOGIC SHIFT UNIT 50

- 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 micooperation, the contents of specified registers are placed in the inputs of the common 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 shift microoperations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.

ARITHMETIC LOGIC SHIFT UNIT

One stage
of an arithmetic logic shift unit



S3	S2	S1	S0	Cin	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 + B'	Subtract with borrow
0	0	1	0	1	F = A + B' + 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	TransferA
0	1	0	0	X	F = A ∧ B	AND
0	1	0	1	X	F = A ∨ B	OR
0	1	1	0	X	F = A ⊕ B	XOR
0	1	1	1	X	F = 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

b) What are the number systems conversions available? Explain with an example. CO1 L2 7M

Number systems are the technique to represent numbers in the computer system architecture, every value that you are saving or getting into/from computer memory has a defined number system.

Computer architecture supports following number systems.

- Binary number system
- Octal number system
- Decimal number system
- Hexadecimal (hex) number system

i) Binary Number System

A Binary number system has only two digits that are 0 and 1. Every number (value) represents with 0 and 1 in this number system. The base of binary number system is 2, because it has only two digits.

For example, $(101101)_2$ in decimal is
$= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 1 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$
$= 32 + 8 + 4 + 1$
$= (45)_{10}$

ii) Octal number system

Octal number system has only eight (8) digits from 0 to 7. Every number (value) represents with 0,1,2,3,4,5,6 and 7 in this number system. The base of octal number system is 8, because it has only 8 digits.

For example, $(24)_8$ in decimal is
$= 2\times8^1+4\times8^0$
$= (20)_{10}$

iii) Decimal number system

Decimal number system has only ten (10) digits from 0 to 9. Every number (value) represents with 0,1,2,3,4,5,6, 7,8 and 9 in this number system. The base of decimal number system is 10, because it has only 10 digits.

For example, the value of 786 is
$= 7 \times 10^2 + 8 \times 10^1 + 6 \times 10^0$
$= 700 + 80 + 6$

iv) Hexadecimal number system

A Hexadecimal number system has sixteen (16) alphanumeric values from 0 to 9 and A to F. Every number (value) represents with 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E and F in this number system. The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here A is 10, B is 11, C is 12, D is 13, E is 14 and F is 15.

For example $(3A)_{16} = (00111010)_2$

To convert from Binary to Hexadecimal, group the bits in groups of 4 and write the hex for the 4-bit binary. Add 0's to adjust the groups.
1111011011
$(001111011011)_2 = (3DB)_{16}$

(OR)

3. a) What are the different ways to implement a common bus system and explain with a neat sketch? **CO1 L2 7M**

- **There are two ways to implement the common BUS System.**

  I) Multiplexers

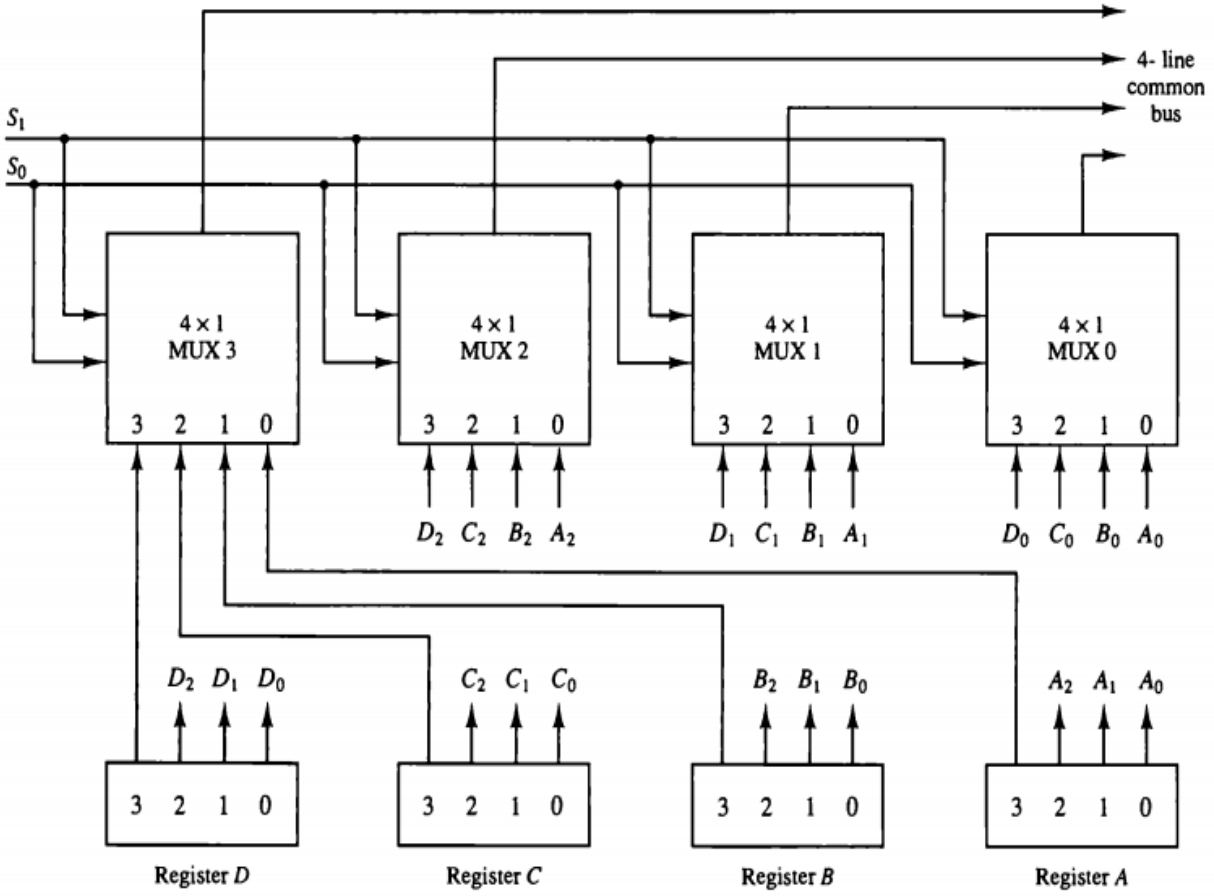
  II) Three state buffer

Bus Transfers

- Paths must be provided to transfer information from one register to another
- A *Common Bus System* is a scheme for transferring information between registers in a multiple-register configuration
- A **bus**: set of common lines, one for each bit of a register, through which binary information is transferred one at a time
  - Bus is a path (of a group of wires) over which information is transferred, from any of several sources to any of several destinations
- Control signals determine which register is selected by the bus during each particular register transfer
- From a register to bus: BUS ← R,

I) Multiplexers:-

One way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus. The construction of a bus system for four registers is shown in Fig. 4-3. Each register has four 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. In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers. For example, output 1 of register A is connected to input 0 of MUX 1 because this input is labeled A_1. The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus. Thus MUX 0 multiplexes the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other two bits.



The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus. When $S_1S_0 = 00$, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers. Similarly, register B is selected if $S_1S_0 = 01$, and so on. Table 4-2 shows the register that is selected by the bus for each of the four possible binary value of the selection lines.

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

In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register. The size of each multiplexer must be $k \times 1$ since it multiplexes k data lines. For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control of the particular destination register selected. The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement. When the bus is includes in the statement, the register transfer is symbolized as follows:

$$BUS \leftarrow C, \qquad R1 \leftarrow BUS$$

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

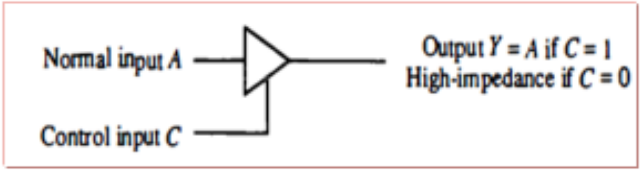
$$R1 \leftarrow C$$

From this statement the designer knows which control signals must be activated to produce the transfer through the bus.

II) Three state buffer:-

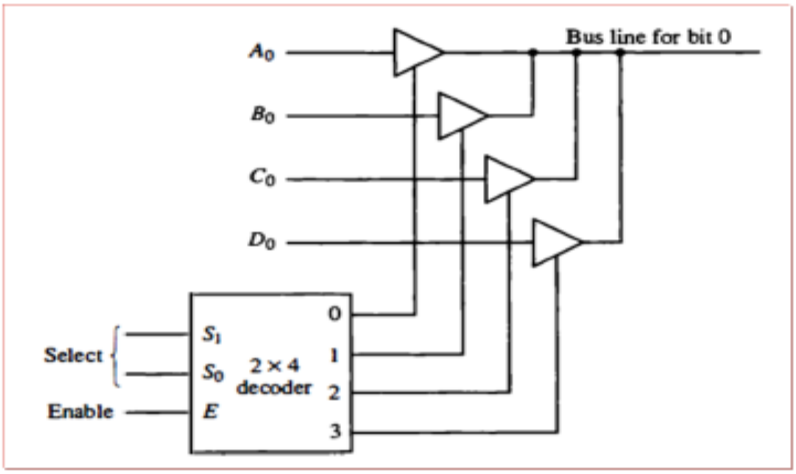
Three-State Bus Buffers

- A bus system can be constructed with three-state buffer gates instead of multiplexers
- A three-state buffer is a digital circuit that exhibits three states: logic-0, logic-1, and high-impedance (Hi-Z)
- The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance.
- Three-state gates may perform any conventional logic, such as AND or NAND.
- However, the one most commonly used in the design of a bus system is the buffer gate.
- Graphic symbols for three state buffer.



Bus line with three state-buffers

- Bus line with three-state buffer (replaces MUX0 in the previous diagram)
- Bus line with three state-buffers.



b) Label the diagram for 4-bit binary adder and 4-bit adder-subtracter. CO1 L3 7M

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

The block diagram for a 4-bit adder-subtractor circuit can be represented as:

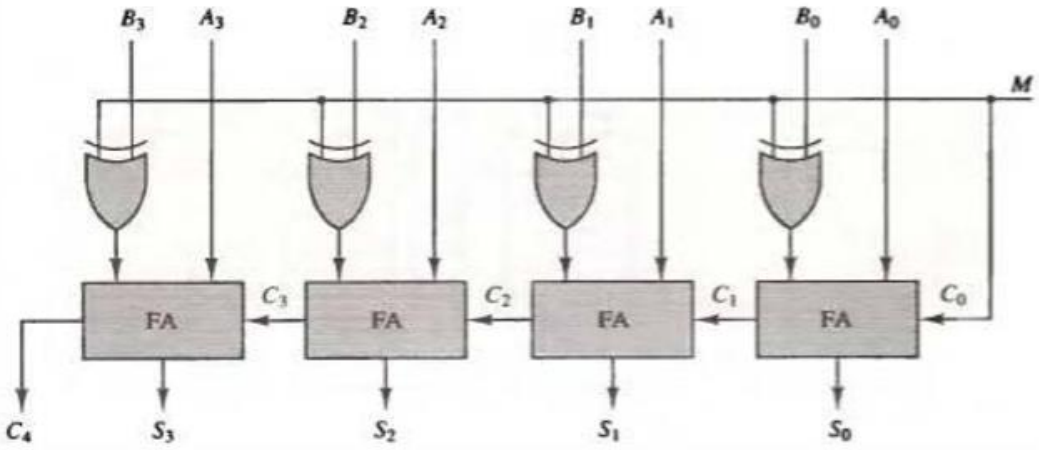


Figure 4-7 4-bit adder-subtractor.

2's complement of B. For unsigned numbers, this gives $A - B$ if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$. For signed numbers, the result is $A - B$ provided that there is no overflow.

- When the mode input (M) is at a low logic, i.e. '0', the circuit act as an adder and when the mode input is at a high logic, i.e. '1', the circuit act as a subtractor.
- The exclusive-OR gate connected in series receives input M and one of the inputs B.
- When M is at a low logic, we have B⊕0 = B. The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
- When M is at a high logic, we have B⊕ 1 = B' and C0 = 1.The B inputs are complemented, and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B.

Unit –II

4. a) Name the registers for the basic computer with number of bits used and describe their functionality. CO2 L3 7M

2. Computer Registers

- It is necessary to provide a register in the control unit for storing the instruction code after it is read from memory.
- The computer needs processor registers for manipulating data and a register for holding a memory address.

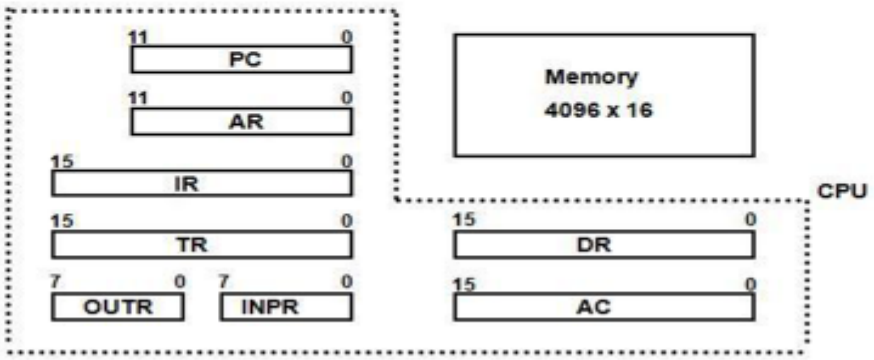


Figure 2.4: Basic Computer Register and Memory

Computer Registers

Register Symbol	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

Computer Registers

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

b) Interpret the symbols and binary code used for microinstruction fields. CO2 L2 7M

- **Microinstructions**
  - Control words stored in control memory
  - Specify control signals for execution of micro operations
- **Microinstruction Fields**

F1	F2	F3	CD	BR	AD
3bits	3 bits	3 bits	2 bits	2 bits	7 bits

F1, F2, F3: Micro operation fields

CD: Condition for branching

BR: Branch field

AD: Address field

Microinstruction Fields

F1	Microoperation	Symbol
000	None	NOP
001	AC ← AC + DR	ADD
010	AC ← 0	CLRAC
011	AC ← AC + 1	INCAC
100	AC ← DR	DRTAC
101	AR ← DR(0-10)	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	AC ← AC - DR	SUB
010	AC ← AC ∨ DR	OR
011	AC ← AC ∧ DR	AND
100	DR ← M[AR]	READ
101	DR ← AC	ACTDR
110	DR ← DR + 1	INCDR
111	DR(0-10) ← PC	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	AC ← AC ⊕ DR	XOR
010	AC ← AC'	COM
011	AC ← shl AC	SHL
100	AC ← shr AC	SHR
101	PC ← PC + 1	INCPC
110	PC ← AR	ARTPC
111	Reserved	

Microinstruction Fields

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

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

Fetch Routine

- Fetch routine
  - Read instruction from memory
  - Decode instruction and update PC

Microinstructions for fetch routine:

```
AR ← PC
DR ← M[AR], PC ← PC + 1
AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0
```

Symbolic microprogram for fetch routine:

```
        ORG 64
FETCH:  PCTAR          U  JMP  NEXT
        READ, INCPC    U  JMP  NEXT
        DRTAR          U  MAP
```

Binary microporgram for fetch routine:

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Symbolic Microinstruction

- **Sample Format** | **Label:** | **Micro-ops** | **CD** | **BR** | **AD** |

- Label — may be empty or may specify symbolic address terminated with colon
- Micro-ops — consists of 1, 2, or 3 symbols separated by commas
- CD — one of {U, I, S, Z}
  - U: Unconditional Branch
  - I: Indirect address bit
  - S: Sign of AC
  - Z: Zero value in AC
- BR — one of {JMP, CALL, RET, MAP}
- AD — one of {Symbolic address, NEXT, empty}

20

Symbolic Microprogram

- **Control memory:** 128 20-bit words
- **First 64 words:** Routines for 16 machine instructions
- **Last 64 words:** Used for other purpose (e.g., fetch routine and other subroutines)
- **Mapping:** OP-code XXXX into 0XXXX00, first address for 16 routines are 0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, ..., 60

Partial Symbolic Microprogram

Label	Microops	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

22

Binary Microprogram

Micro Routine	Address		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

23

(OR)

5. a) State the phases of an instruction cycle? Design the flowchart for instruction cycle

CO2 L1 7M

In the basic computer each instruction cycle consists of the following phases:

- Fetch an instruction from memory.
- Decode the instruction.
- Read the effective address from memory if the instruction has an indirect address.
- Execute the instruction.

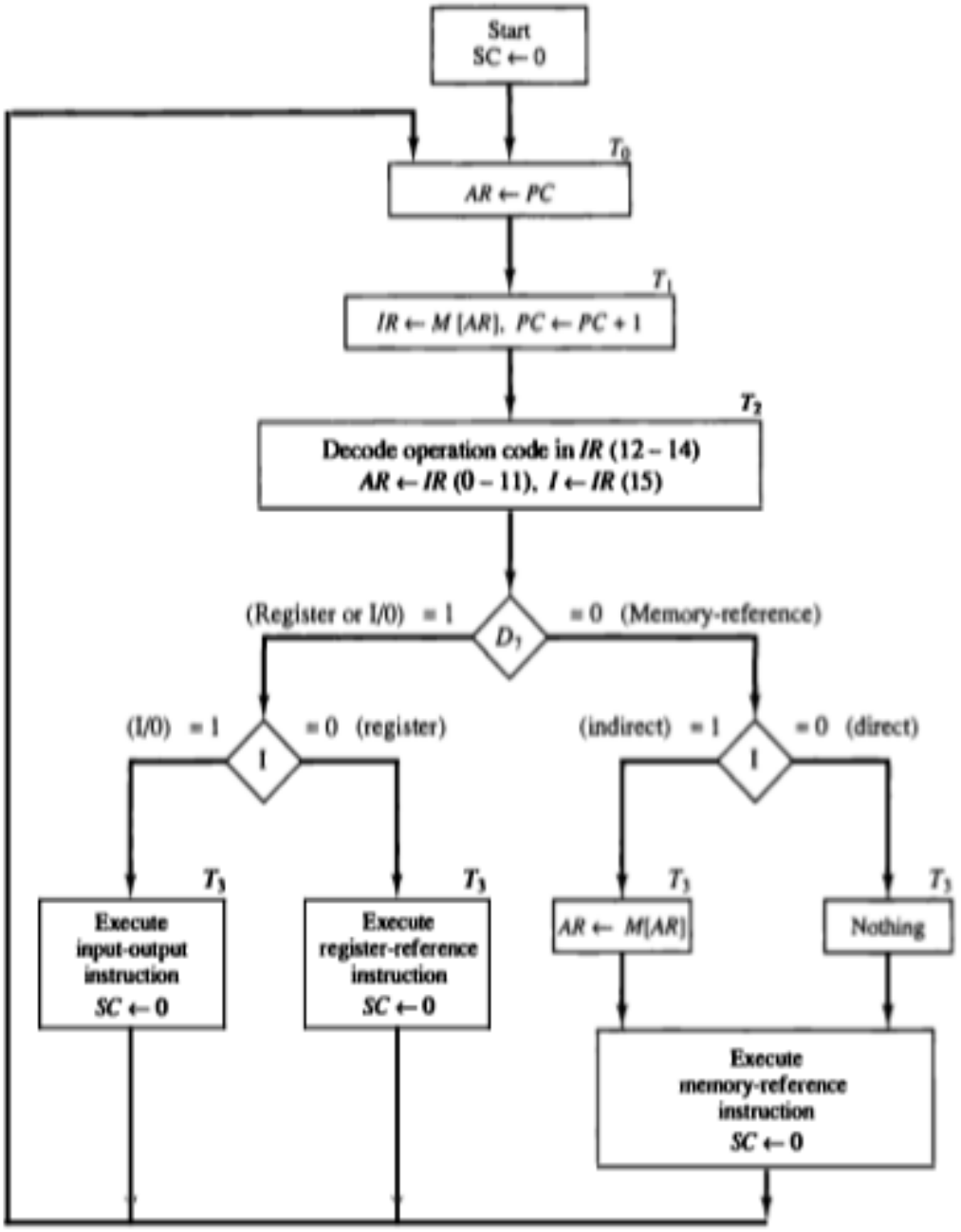


Figure 5-9 Flowchart for instruction cycle (initial configuration).

b) Show the block diagram of the microprogram sequencer and discuss. CO2 L2 7M

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.

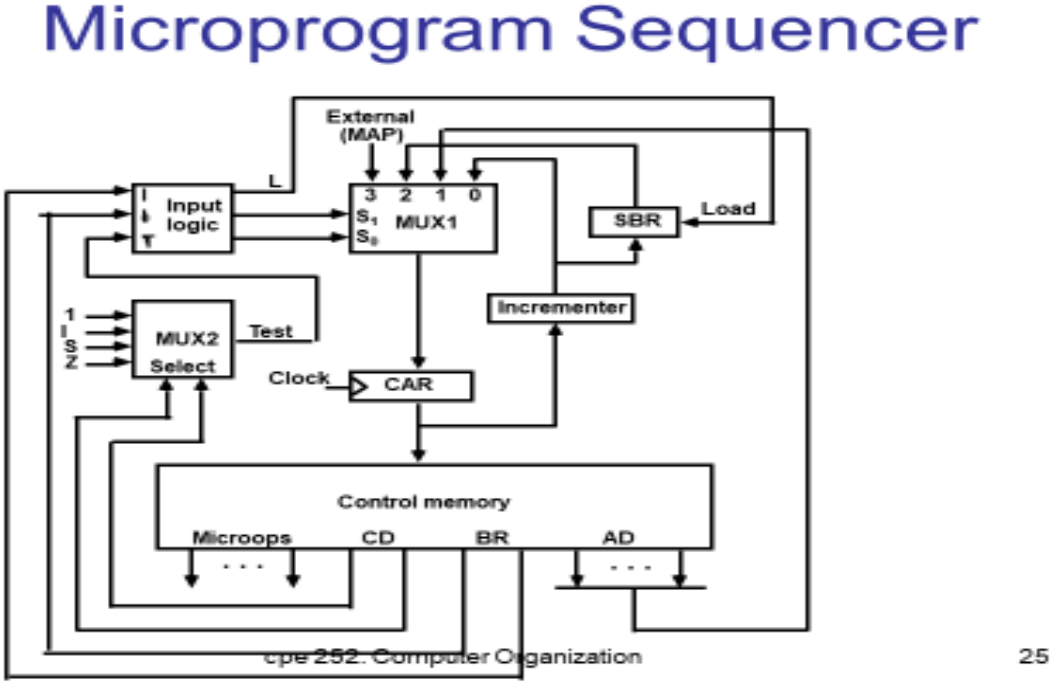
- These are the different ways a microprogram sequencer select the address.

– Incrementing CAR

– Unconditional or conditional branch, depending on status bit conditions

– Mapping from bits of instruction to address for control memory

– Facility for subroutine call and return



The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

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

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	x	1	0	0
1	1	1	1	x	1	1	0

Fig:- Input Logic Truth table for Microprogram Sequencer

Unit –III

6. a) Examine the procedure involved in reverse polish notation with an example. **CO3 L3 7M**

Reverse Polish Notation (RPN) was devised as a method of simplifying mathematical Expressions.

REVERSE POLISH NOTATION

Arithmetic Expressions: A + B

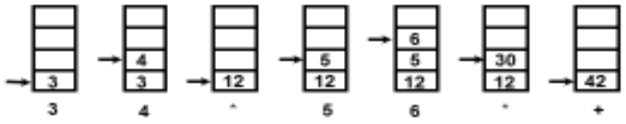
A + B	Infix notation
+ A B	Prefix or Polish notation
A B +	Postfix or reverse Polish notation

- The reverse Polish notation is very suitable for stack manipulation

Evaluation of Arithmetic Expressions

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

(3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +



b) Inspect the hardware for signed-magnitude addition and subtraction. CO3 L2 7M

Addition and Subtraction

Addition and Subtraction
- Sign-magnitude
- 2's complement

Addition and Subtraction with Signed-Magnitude Data

TABLE 10-1 Addition and Subtraction of Signed-Magnitude Numbers

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

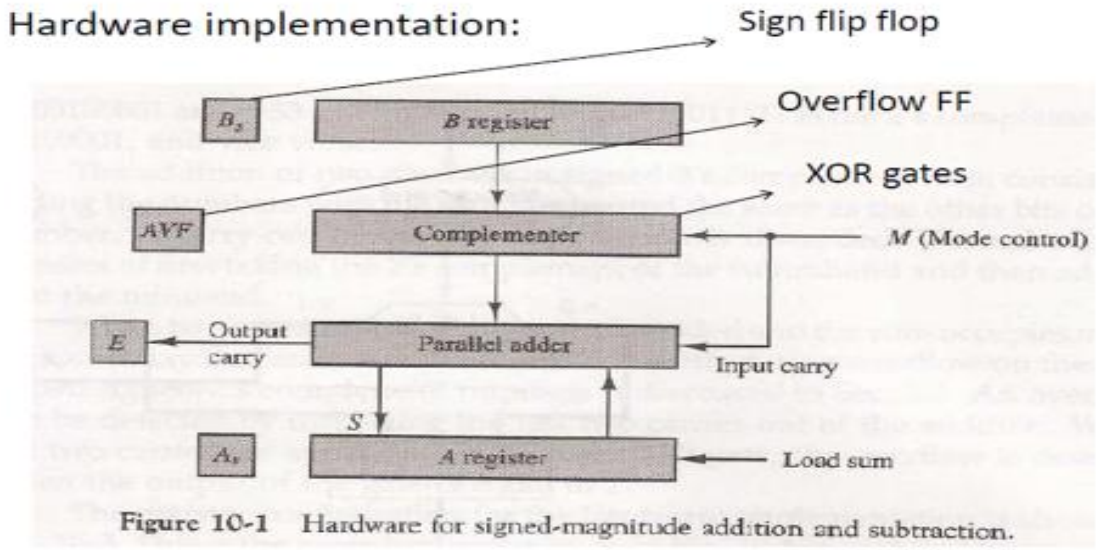


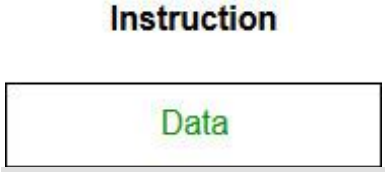
Figure 10-1 Hardware for signed-magnitude addition and subtraction.

(OR)

7. a) List out any seven addressing modes and interpret each addressing mode with syntax. CO3 L2 7M

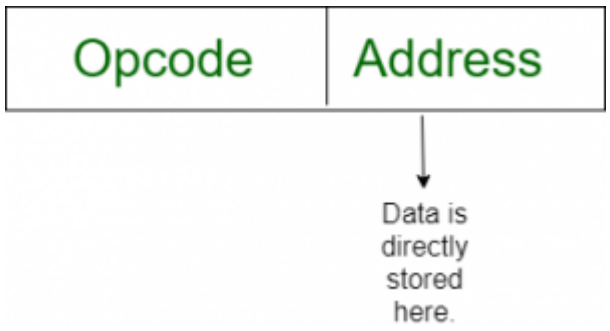
Addressing Modes– The term addressing modes refers to the way in which the operand of an instruction is specified.

- **Implied mode:**: In implied addressing the operand is specified in the instruction itself. In this mode the data is 8 bits or 16 bits long and data is the part of instruction.Zero address instruction are designed with implied addressing mode.



Example: CLC (used to reset Carry flag to 0)

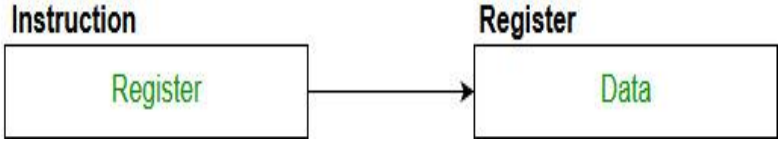
- **Immediate addressing mode (symbol #):**In this mode data is present in address field of instruction .Designed like one address instruction format.
Note: Limitation in the immediate mode is that the range of constants are restricted by size of address field.



Example: MOV AL, #35H (move the data 35H into AL register)

- **Register mode:** In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers. The data is in the register that is specified by the instruction.

Here one register reference is required to access the data.



Example: MOV AX,CX (move the contents of CX register to AX register)

- **Register Indirect mode**: In this addressing the operand's offset is placed in any one of the registers BX,BP,SI,DI as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction.
  Here two register reference is required to access the data.

The 8086 CPUs let you access memory indirectly through a register using the register indirect addressing modes.

- MOV AX, [BX](move the contents of memory location s

addressed by the register BX to the register AX)

- **Auto Indexed (increment mode)**: Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.**(R1)+**.
  Here one register reference,one memory reference and one ALU operation is required to access the data.
  Example:

- Add R1, (R2)+ // OR
- R1 = R1 +M[R2]

R2 = R2 + d

Useful for stepping through arrays in a loop. R2 – start of array d – size of an element

- **Auto indexed (decrement mode)**: Effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. –**(R1)**
  Here one register reference,one memory reference and one ALU operation is required to access the data.

Example:

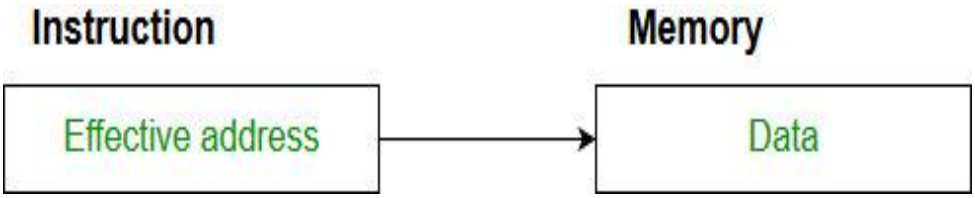
Add R1,-(R2) //OR

R2 = R2-*d*

R1 = R1 + M[R2]

Auto decrement mode is same as auto increment mode. Both can also be used to implement a stack as push and pop . Auto increment and Auto decrement modes are useful for implementing "Last-In-First-Out" data structures.

- **Direct addressing/ Absolute addressing Mode (symbol [ ]):** The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element. In this addressing mode the 16 bit effective address of the data is the part of the instruction.
  Here only one memory reference operation is required to access the data.



Example:ADD AL,[0301] //add the contents of offset address 0301 to AL

- **Indirect addressing Mode (symbol @ or ())**:In this mode address field of instruction contains the address of effective address.Here two references are required.
  1st reference to get effective address.
  2nd reference to access the data.
  Based on the availability of Effective address, Indirect mode is of two kind:

  1. Register Indirect:In this mode effective address is in the register, and corresponding register name will be maintained in the address field of an instruction.
     Here one register reference,one memory reference is required to access the data.
  2. Memory Indirect:In this mode effective address is in the memory, and corresponding memory address will be maintained in the address field of an instruction.
     Here two memory reference is required to access the data.

- **Indexed addressing mode**: The operand's offset is the sum of the content of an index register SI or DI and an 8 bit or 16 bit displacement.
  Example:MOV AX, [SI +05]

- **Based Indexed Addressing:** The operand's offset is sum of the content of a base register BX or BP and an index register SI or DI.
  Example: ADD AX, [BX+SI] .

Based on Transfer of control, addressing modes are:

- **PC relative addressing mode:** PC relative addressing mode is used to implement intra segment transfer of control, In this mode effective address is obtained by adding displacement to PC.
- EA= PC + Address field value

PC= PC + Relative value.

- **Base register addressing mode:**Base register addressing mode is used to implement inter segment transfer of control.In this mode effective address is obtained by adding base register value to address field value.
- EA= Base register + Address field value.

PC= Base register + Relative value.

b) Display the flowchart for Booth multiplication operation and discuss the operations performed. CO3 L4 7M

Booth Multiplication Algorithm

Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.
It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

For example, the binary number 001 110 (+ 14) has a string of 1's from 2^3 to 2^1 (k = 3, m = 1).
The number can be represented as $2^{k+1} - 2^m = 2^4 - 2^1 = 16 - 2 = 14$.

Therefore, the multiplication M x 14, where M is the multiplicand and 14 the multiplier, can be done as M x 2^4 - M X 2^1

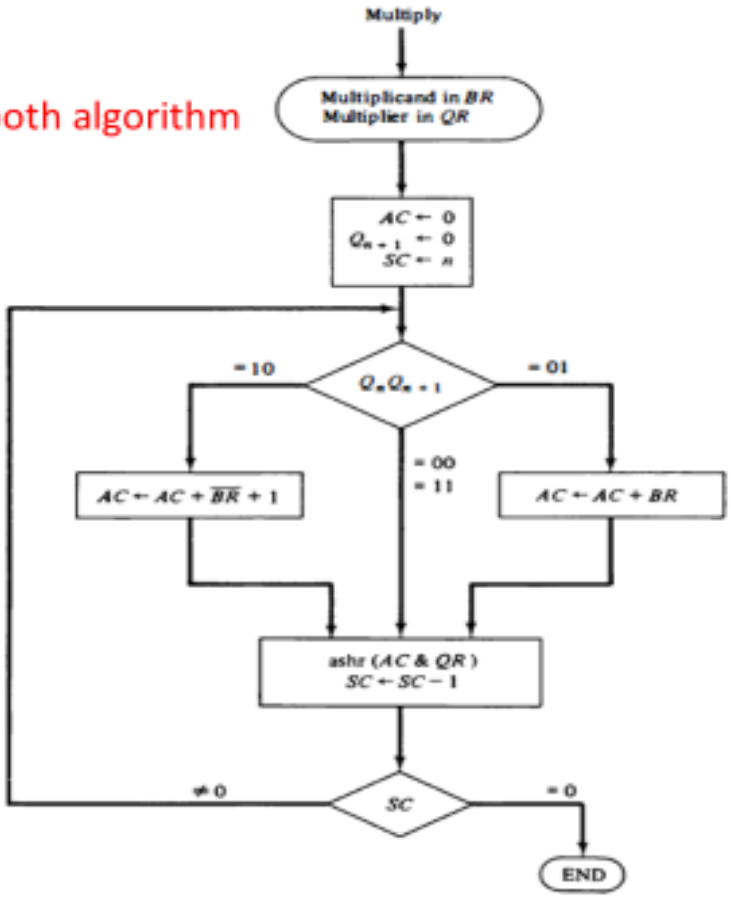
A= 00011 B= 00111 => A*B= A*(7)=A* (8-1)=A*8-A*1

Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

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.

Flowchart for Booth algorithm



8. a) Examine the working of associate memory with a neat diagram. CO4 L2 7M

The block diagram of an associative memory is shown in Fig. 12-6. It consists of a memory array and logic for m words with n bits per word. 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 the fact 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 1's. Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared. Thus the key provides a mask or identifying piece of information which

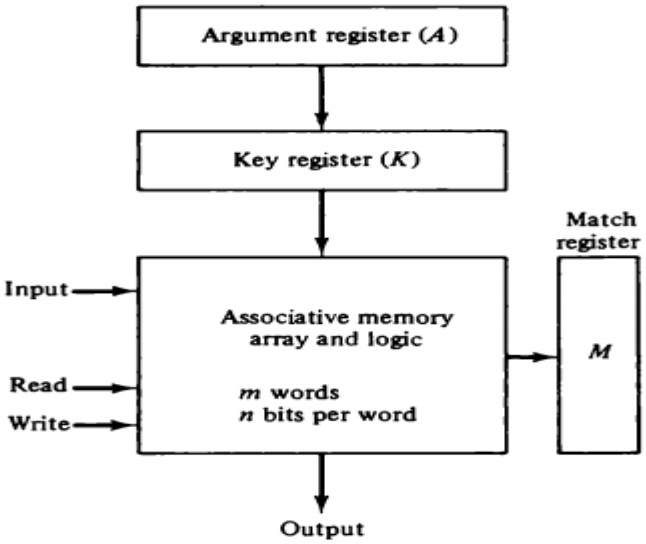


Fig: Block diagram of Associative memory

specifies how the reference to memory is made. To illustrate with a numerical example, suppose that the argument register 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 Fig. 12-7. The cells in the array are marked by the letter C with two subcripts. 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. A 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 $j = 1, 2, \ldots, n$. 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.

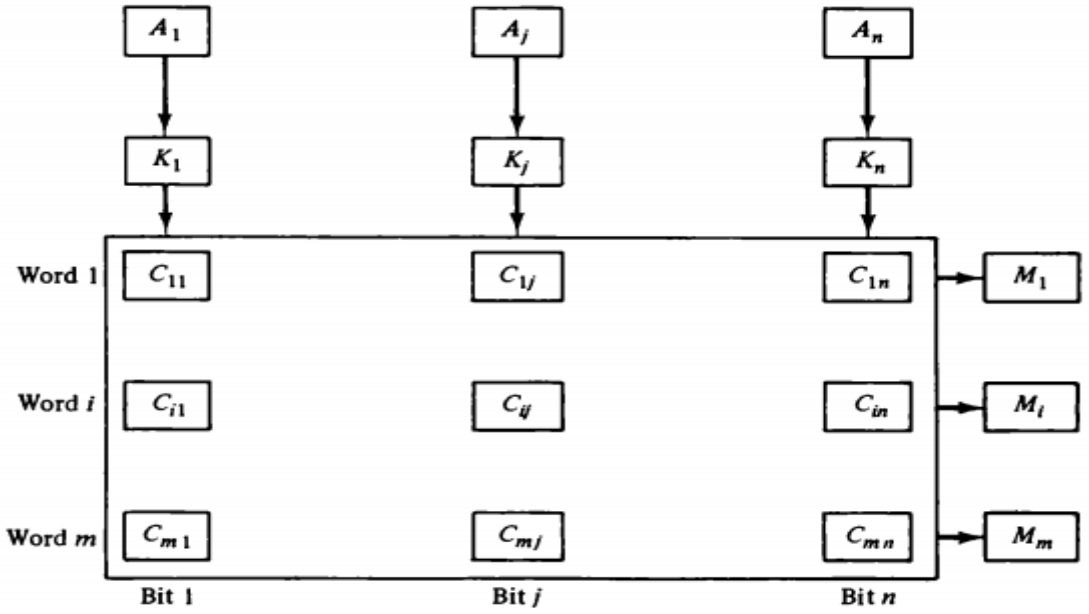
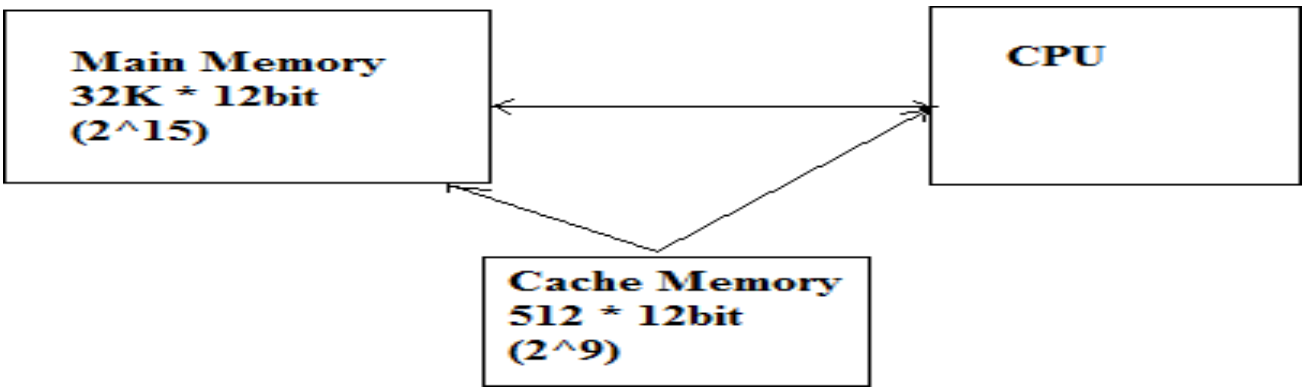


Fig: Associative memory of m words, n cells per word.

b) Illustrate the mapping procedures while considering the organization of cache memory. **CO4 L3 7M**

- 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 cache memory
- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU component.
- When CPU needs to access memory, the cache is examined
- If the word is found in the cache, it is read from the fast memory
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**
- Otherwise, it is a **miss**
- The performance of cache memory is frequently measured in terms of aquantity called **hit ratio**
- **Hit ratio = hit / (hit+miss**
- The basic characteristic of cache memory is its fast access time.

- Therefore, very little or no time must be wasted when searching the words in the cache
- The transformation of data from main memory to cache memory is referred to as a **mapping** process, there are three types of mapping:
  - **Associative mapping**
  - **Direct mapping**
  - **Set-associative mapping**
-



Associative mapping

- The fastest and most flexible cache organization uses an associative memory
- The associative memory stores both the address and data of the memory word
- This permits any location in cache to store ant word from main memory
- 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
- A CPU address of 15 bits is places in the argument register and the associative memory us searched for a matching address
- If the address is found, the corresponding 12- bits data is read and sent to the CPU
- If not, the main memory is accessed for the word
- If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache.

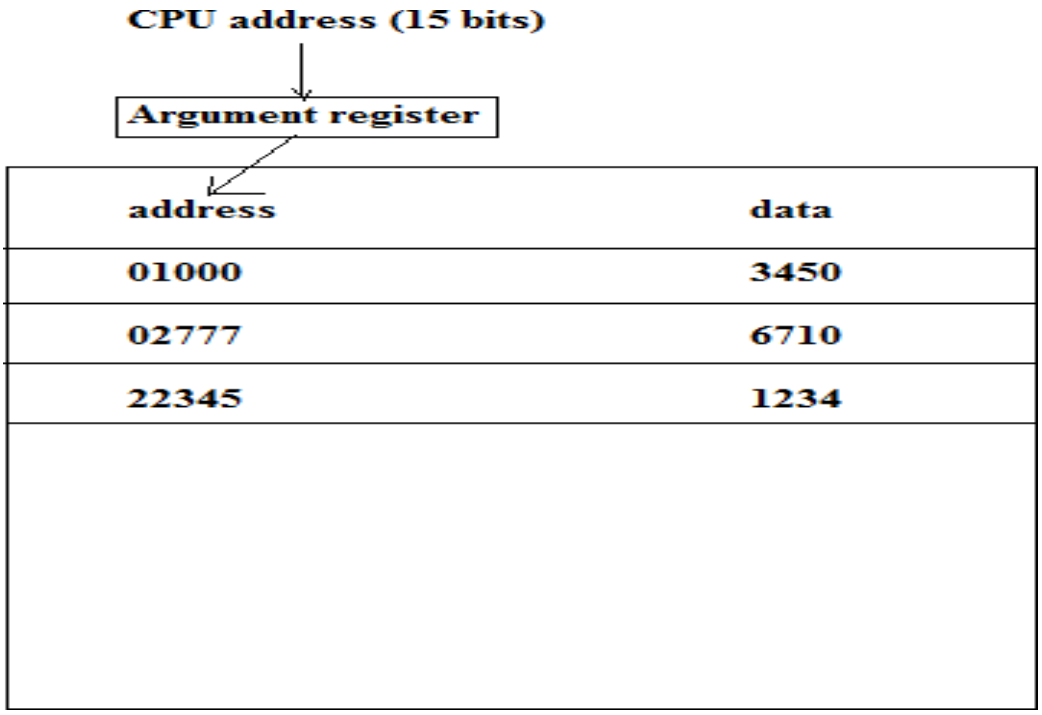


Fig: - Associative mapping cache

Direct Mapping

- Associative memory is expensive compared to RAM
- In general case, there are 2^k words in cache memory and 2^n words in main memory (in our case, k=9, n=15)
- The n bit memory address is divided into two fields: k-bits for the index and n-k bits for the tag field,

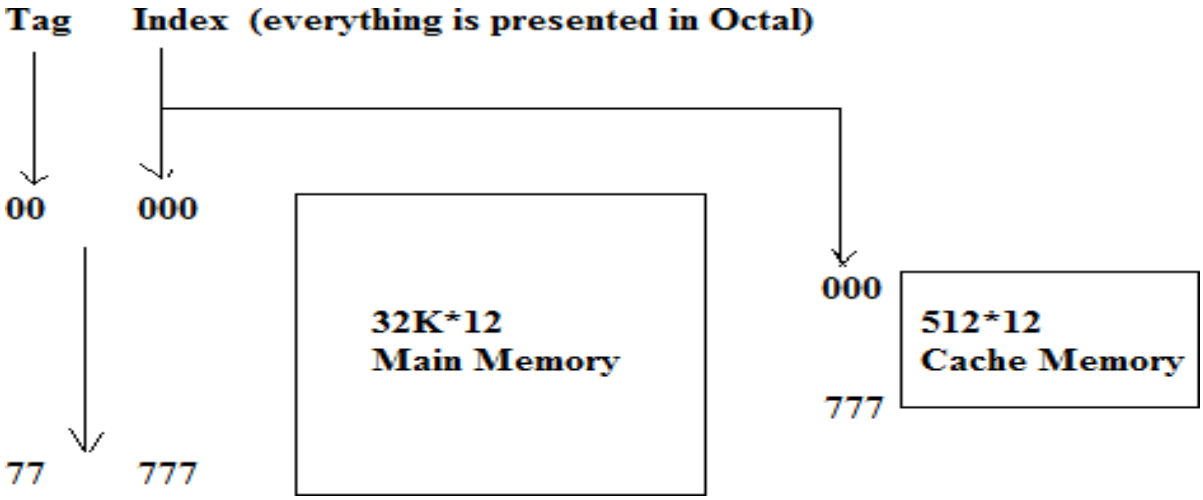


Fig: - Addressing relationships between main and cache memories.

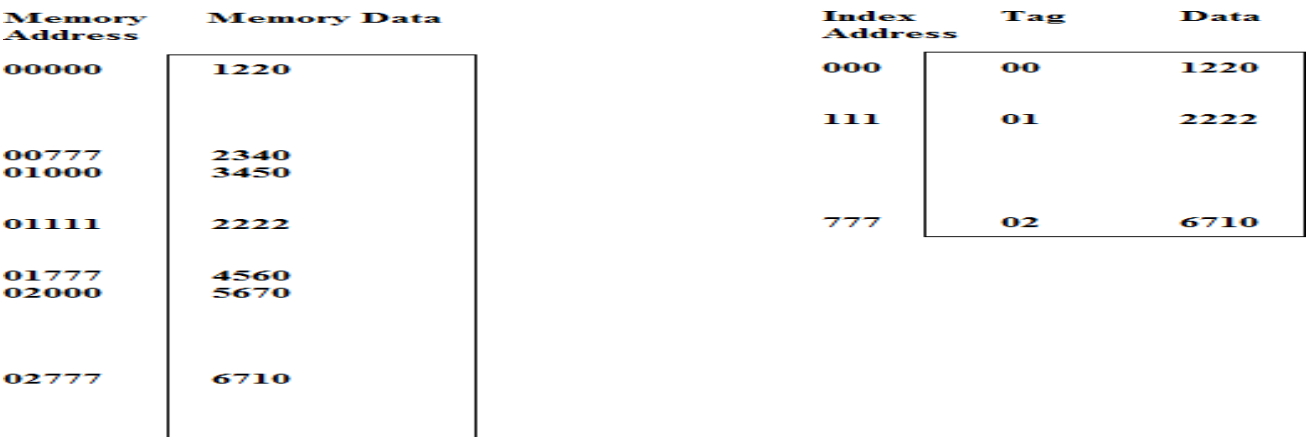


Fig: - Direct Mapping cache organization.

Set-Associative Mapping

- The disadvantage of direct mapping is that 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 an improvement over the direct-mapping in that each word of cache can store two or more word of memory under the same index address.
- Each index address refers to two data words and their associated tags
- Each tag requires six bits and each data word has 12 bits, so the word length is 2*(6+12) = 36 bits.

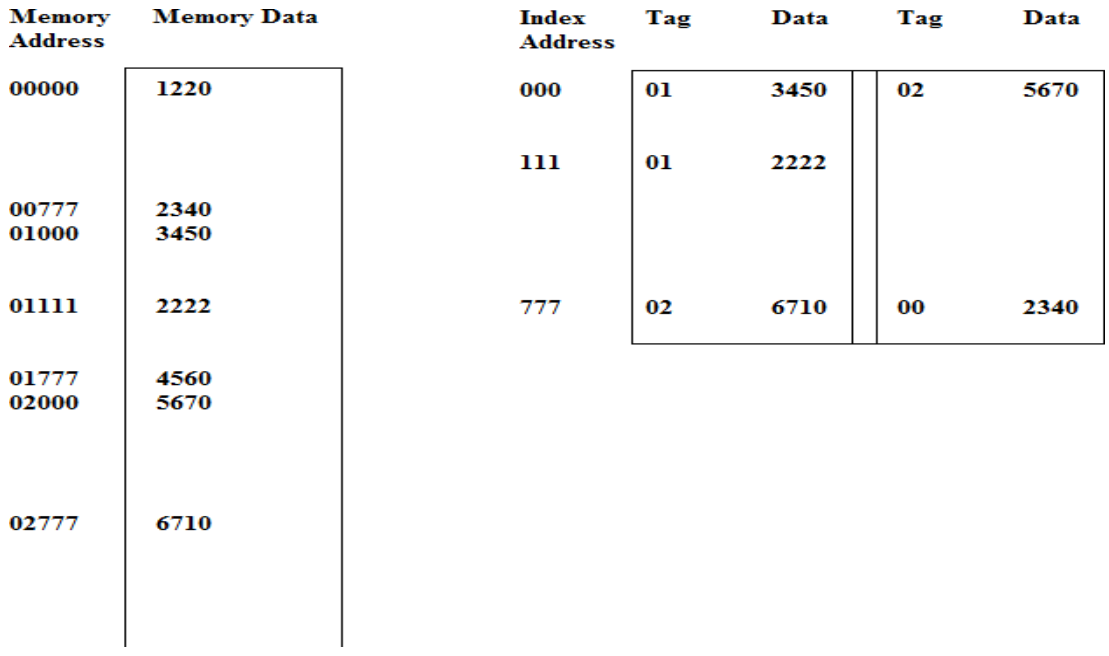


Fig:- Two-way set-associative mapping cache.

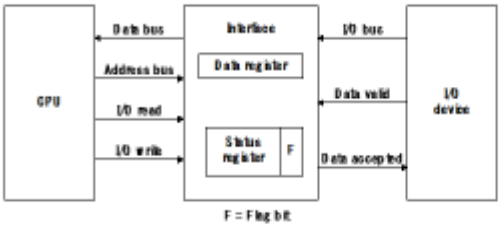
(OR)

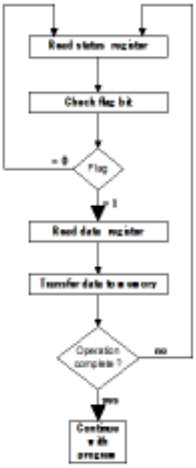
9. Analyse the various modes of data transfer to and from peripherals CO4 L2 14M

11-1

11-4 Modes of Transfer

- Data transfer to and from peripherals
  - 1) Programmed I/O : *in this section*
  - 2) Interrupt-initiated I/O : *in this section and sec. 11-5*
  - 3) Direct Memory Access (DMA) : *sec. 11-6*
  - 4) I/O Processor (IOP) : *sec. 11-7*
- Example of Programmed I/O : *Fig. 11-10, 11-11*





- Interrupt-initiated I/O
  - 1) Non-vectored : fixed branch address
  - 2) Vectored : interrupt source supplies the branch address (interrupt vector)

11-25

- Software Considerations
  - I/O routines
    - » software routines for controlling peripherals and for transfer of data between the processor and peripherals
  - I/O routines for standard peripherals are provided by the manufacturer (Device driver, OS or BIOS)
  - I/O routines are usually included within the operating system
  - I/O routines are usually available as operating system procedures (OS or BIOS function call)

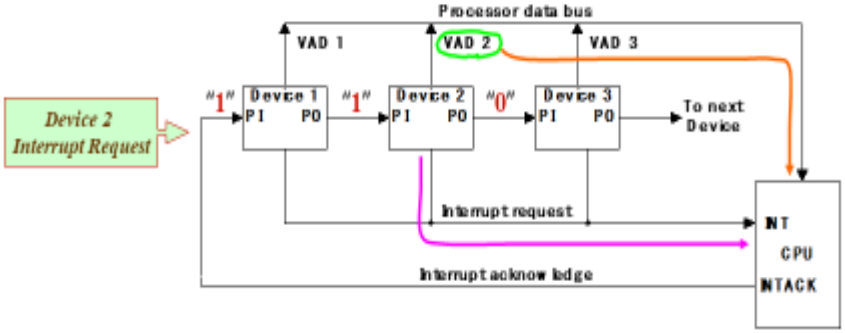
11-5 Priority Interrupt

- Priority Interrupt
  - Identify the source of the interrupt when several sources will request service simultaneously
  - Determine which condition is to be serviced first when two or more requests arrive simultaneously
    - » 1) Software : Polling
    - » 2) Hardware : Daisy chain, Parallel priority

◆ Polling

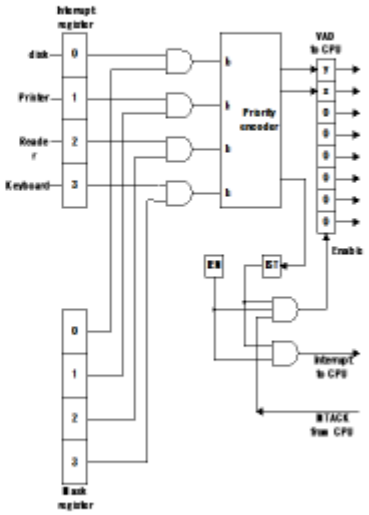
- Identify the highest-priority source by software means
  - » One common branch address is used for all interrupts
  - » Program polls the interrupt sources in sequence
  - » The highest-priority source is tested first
- Polling priority interrupt If there are many interrupt sources, the time required to poll them can exceed the time available to service the I/O device
  - » Hardware priority interrupt

  Daisy-Chaining : ***Fig. 11-12***



◆ Parallel Priority

- Priority Encoder –
- Parallel Priority : ***Fig. 11-14***
  - » Interrupt Enable F/F (**IEN**) : set or cleared by the program
  - » Interrupt Status F/F (**IST**) : set or cleared by the encoder output
- Priority Encoder Truth Table : ***Tab. 11-2***



◆ Interrupt Cycle

- At the end of each instruction cycle, CPU checks IEN and IST
- if both IEN and IST equal to "1"
- CPU goes to an Instruction Cycle
  - » Sequence of microoperation during Instruction Cycle

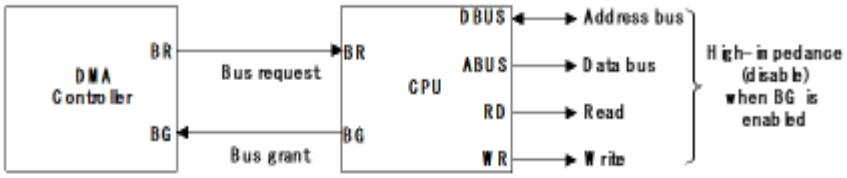
Branch to ISR

$SP \leftarrow SP - 1$	: Decrement stack point
$M[SP] \leftarrow PC$	: Push PC into stack
$INTACK \leftarrow 1$	: Enable INTACK
$PC \leftarrow VAD$	: Transfer VAD to PC
$IEN \leftarrow 0$	: Disable further interrupts
Go to Fetch next instruction	

11-6 Direct Memory Access (DMA)

DMA

- DMA controller takes over the buses to manage the transfer directly between the **I/O device** and **memory** (**Bus Request/Grant**)

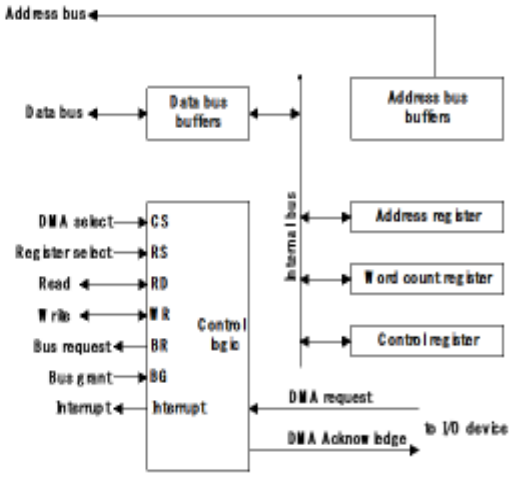


Transfer Modes

- 1) Burst transfer : Block
- 2) Cycle stealing transfer : Byte

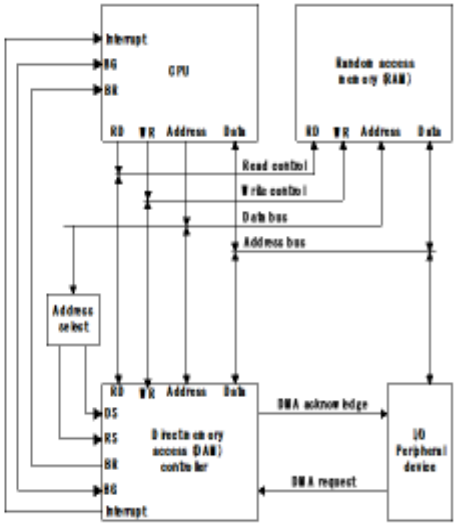
DMA Controller (Intel 8237 DMAC) : *Fig. 11-17*

- DMA Initialization Process
  - » 1) Set Address register :
    - memory address for read/write
  - » 2) Set Word count register :
    - the number of words to transfer
  - » 3) Set transfer mode :
    - read/write,
    - burst/cycle stealing,
    - I/O to I/O,
    - I/O to Memory,
    - Memory to Memory
    - Memory search
    - I/O search
  - » 4) DMA transfer start : *next section*
  - » 5) EOT (End of Transfer) :
    - Interrupt



◆ DMA Transfer (I/O to Memory)

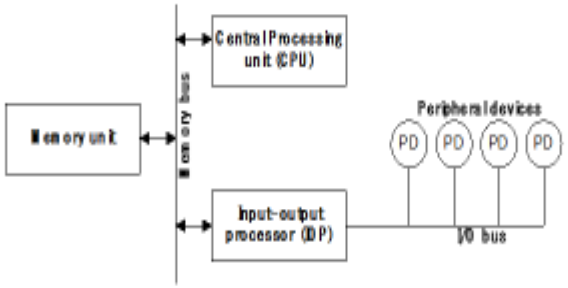
- 1) I/O Device sends a DMA request
- 2) DMAC activates the **BR** line
- 3) CPU responds with **BG** line
- 4) DMAC sends a DMA acknowledge to the I/O device
- 5) I/O device puts a word in the data bus (*for memory write*)
- 6) DMAC write a data to the address specified by **Address register**
- 7) Decrement **Word count register**
- 8) **Word count register** **EOT** interrupt CPU
- 9) **Word count register** DMAC checks the DMA request from I/O device



■ 11-7 Input-Output Processor (IOP)

◆ IOP :

- Communicate directly with all I/O devices
- Fetch and execute its own instruction
  - » IOP instructions are specifically designed to facilitate I/O transfer
  - » DMAC must be set up entirely by the CPU
- Designed to handle the details of I/O processing



◆ Command

- **Instruction** *that are read form memory by an* ***IOP***
  - » Distinguish from instructions that are read by the CPU
  - » Commands are prepared by experienced programmers and are stored in memory
  - » Command word = IOP program

◆ **CPU - IOP Communication :**

l **Memory units acts as a message center : Information**

» **each processor leaves information for the other**

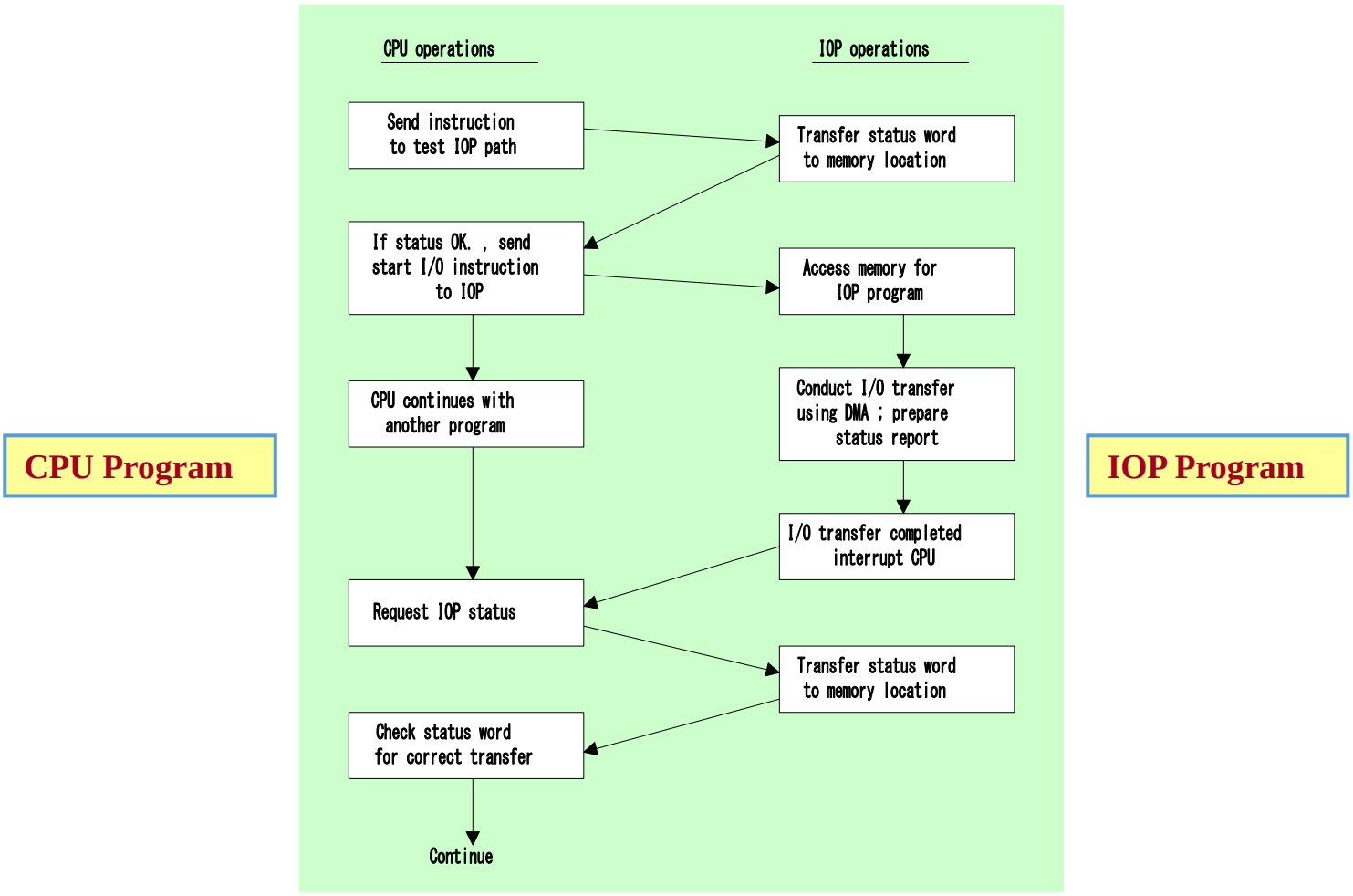


Fig:- CPU - IOP Communication

Scheme prepared by

(R.VEERAMOHANA RAO)

Signature of HOD

SIGNATURE OF EVALUATORS.