GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN (AUTONOMOUS)

(Affiliated to Andhra University, Visakhapatnam) II B.Tech. -I Semester Regular Examinations, Nov – 2025

Subject: COMPUTER ORGANIZATION (24CT11RC11) Date:21-11-2025

QUESTION PAPER KEY

1 a) Define micro-operation and explain the four Basic types of shift micro-operation and their variants. 7M

Ans:

Micro-operation: It is an operation executed on data stored in registers. Basic operations on which more complex instructions are built – each execution phase (e.g., fetch) consists of one or more sequential micro-ops. Each micro-op is executed in one clock cycle in some subsection of the processor circuitry CPU. The System cycle time determined by longest micro-op. Many micro-ops (for successive instructions) can be executed simultaneously – if non-conflicting, independent areas of circuitry.

Shift Micro-operations:

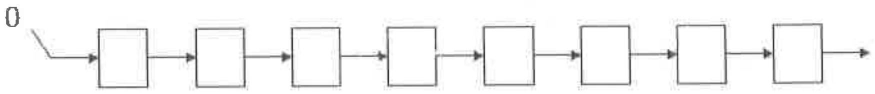
Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations. The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input. During a shift-left operation the serial input transfers a bit into the rightmost t position. During a shift-right operation the serial input transfers a bit into the leftmost position. The information transferred through the serial input determines the type of shift.

- There are three types of shifts(What differentiates them is the information that goes into the serial input)

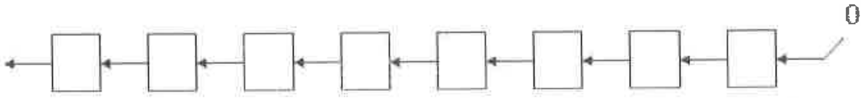
1.Logical shift *2.Circular shift* *3. Arithmetic shift*

1. LOGICAL SHIFT: In a logical shift the serial input to the shift is a 0.

- A right logical shift operation: *shr* for a logical shift right



- A left logical shift operation: *shl* for a logical shift left



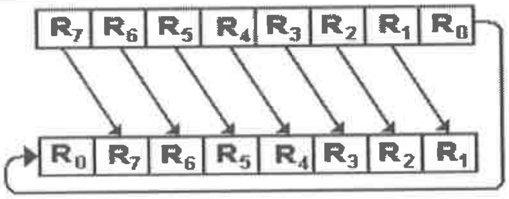
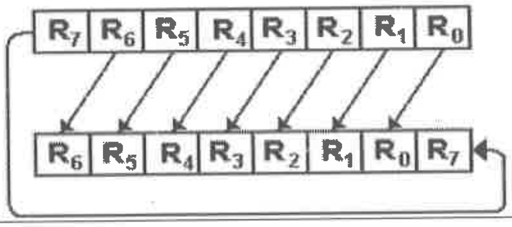
Example:

If R2 = 11001001

Result is stored in R1	Micro operation
10010010	R1 ← shl R2
01100100	R1 ← shr R2

2. CIRCULAR SHIFT:

In a circular shift the serial input is the bit that is shifted out of the other end of the register.

A right circular shift operation *cir* for a circular shift right	A left circular shift operation *cil* for a circular shift left
	

Example:

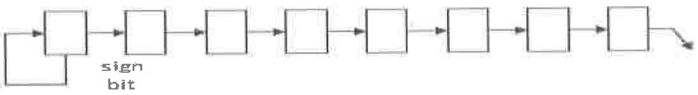
If R2 = 11001001

Result is stored in R1	Micro operation
11100100	R1 ← cir R2
10010011	R1 ← cil R2

3. ARITHMETIC SHIFT: An arithmetic shift is meant for signed binary numbers (integer).

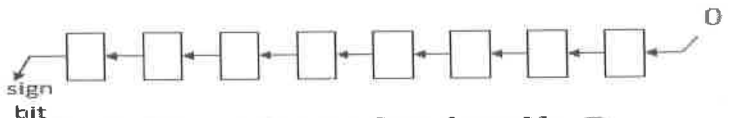
- A right arithmetic shift operation: ashr for arithmetic shift right

  An arithmetic right shift divides a signed number by two.



- A left arithmetic shift operation: ashl for arithmetic shift left

  An arithmetic left shift multiplies a signed number by two



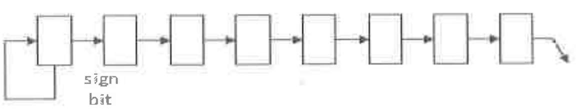
Initial R_{n-1} is lost and replaced by R_{n-2}

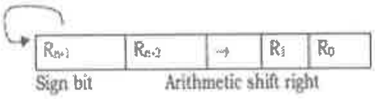
- The main distinction of an arithmetic shift is that it must keep the sign of the number the same as it performs the multiplication or division .An arithmetic shift left inserts a 0 into R0, shifts all other bits to the left. Initial R_{n-1} is lost and replaced by R_{n-2}. A sign reversal is occurs if the bit in R_{n-1} changes in value after the shift after the shift. This happens if the multiplication by 2 causes an overflow.
- Detecting Overflow: overflow occurs if, before the shift, $R_{n-1} \neq R_{n-2}$. An overflow flip-flop V_s is used to detect overflow. $V_s = R_{n-1} \oplus R_{n-2}$
- If V_S =0, there is no overflow, but if Vs= 1, there is an overflow and a sign reversal after the shift. Vs must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

1 b) A computer uses a 16-bit register R that stores the signed binary number 1100110011001100. Determine the value of R in decimal after it undergoes an Arithmetic Shift Right operation. Also, state the condition under which an Arithmetic Shift Left operation causes an overflow. 7M

Ans:

An arithmetic shift is a micro operation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2. An arithmetic shift-right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form.





An overflow occurs after an arithmetic shift left if initially, before the shift, Rn- 1 is not equal to Rn- 2. An overflow flip-flop Vs can be used to detect an arithmetic shift-left overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$

If Vs= 0, there is no overflow, but if Vs 1, there is an overflow and a sign reversal after the shift. Vs must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

Given: A 16-bit signed (two's complement) value stored in register R = 1100 1100 1100 1100_2

Step 1: Convert to decimal (two's complement)

Since the MSB = **1**, the number is **negative**.

Step1: To find its decimal value:

Invert bits:	Add 1:	convert to decimal:
1100110011001100 0011001100110011	0011001100110011 + 1 ------------------------- 0011001100110100	= 0x3334 (hex) = 3×4096+3×256+3×16+4=12288+768+48+4=13108 So original value is: R=−13108

Step 2: Perform Arithmetic Shift Right (ashr): ASR keeps the sign bit (1) and shifts right.

Before ashr	R:1100 1100 1100 1100
After ashr	R: 1110 0110 0110 0110

Convert shifted value to decimal:

Invert bits:	Add 1:	convert to decimal:
1110 0110 0110 0110 0001 1001 1001 1001	0001 1001 1001 1001 + 1 -------------------------------- 0001 1001 1001 1010	Hex = **0x199A** =1×4096+9×256+9×16+10=4096+2304+144+10=6554 The shifted result is: value is: RASR=-6554

Step 3: Detecting Overflow while Performing Arithmetic Shift Left (ashl):

Detect Overflow condition for Arithmetic Shift Left: If shifting left changes the sign of the number, an overflow occurs..

Before ashr	R:1100 1100 1100 1100
After ashr	R: 1001 1001 1001 1000

Check FOR OVERFLOW:

Before ASL: MSB = **1** (negative number)

After ASL: MSB = **1**(negative number)

Therefore: NO overflow.

2. Describe the function of Register Transfer Language (RTL). Write the RTL statements for the following operations:

1. The addition of the contents of registers R2 and R3, storing the result in R1.

2. A conditional transfer: If the input K=1, transfer the contents of R1 to R0. **14M**

Ans:

Resisters –It is a collection of binary storage flip-flops organized in some logical fashion.

Register Transfer Operations

The movement of data stored in registers and the processing performed on the data. The basic components are the set of registers and their function, the binary coded information in the registers, Micro-Operations (performed on the binary data stored in the registers) and Control(The control that initiates the sequence of microoperations).Register transfer Language Can be used to describe what a machine does (an abstract RTN) without describing how the machine does it. Rather than specifying a digital system in words, a specific notation is used, *register transfer language. The symbolic notation used to describe the micro-operation transfers among registers.*

Register Nomenclature:

▪ Letters and numbers – denotes a register (ex. R2, PC, IR)
▪ Parentheses ( ) – denotes a range of register bits (ex. R1(1), PC(7:0), AR(L))
▪ Arrow (←) – denotes data transfer (ex. R1 ← R2, PC(L) ← R0)
▪ Comma – separates parallel operations
▪ Brackets [ ] – Specifies a memory address (ex. R0 ← M[AR], R3 ← M[PC])
▪ A Colon : is used to terminate the control condition

Register transfer Example:

- Information transfer from one register to another is represented as:

  Data Path: $R2 \leftarrow R1$

  Means: transfer the content of register R1 into register R2. R2 content will be replaced by R1 content , during one clock pulse. R1 content will not change.

- Normally transfer occur only under a control condition : $\textit{If } (P = 1) \textit{ then } (R2 \leftarrow R1)$

  Where p, is a control signal. Control function: $P\colon \; R2 \leftarrow R1$

  Arrows denote a transfer of information and the transfer direction. Right-hand side of RTN: always denotes a value Left-hand side of RTN: the name of a location where the value is to be placed (by overwriting the old contents). A Colon is used to terminate the control condition.

1. The addition of the contents of registers R2 and R3, storing the result in R1.	**2. A conditional transfer: If the input K=1, transfer the contents of R1 to R0.**
Sol: **R1 ← R2 + R3** Explanation: • Read the numeric values currently stored in registers R2 and R3. • Add those two values (integer or floating, depending on instruction). • Store the sum into register R1, overwriting whatever was there before.	Sol: **K: R0 ← R1** Explanation: • Read the current value stored in register **R1,** • Copy that value into register **R0,** overwriting whatever was previously in **R0.** • It is a conditional transfer statement. If the input K=1, only then the transfer takes place, else the transfer doesn't takes place.

3 a) Explain the function of the Timing and Control Unit in a basic computer. With a neat flowchart, explain the different phases that constitute the Instruction Cycle. 7M

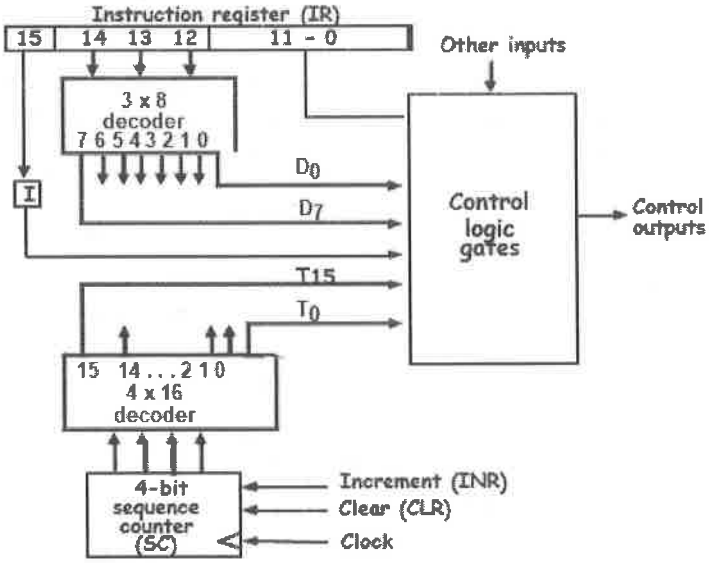
Ans:

Inputs to the control unit come from IR where an instruction is stored.

A hardwired control logic is implemented using inputs:

- A 3x8 decoder to decode opcode bits 12-14 into signals $D_0, \ldots, D_7$
- A flip-flop (I) to store the addressing mode bit in IR
- Address bits from the instruction code (0....11)
- A 4-bit binary sequence counter (SC) to count from 0 to 15 to achieve time sequencing along with INR, CLR (CLR=1 at power-on)
- A 4x16 decoder to decode the output of the counter into 16 timing signals, T0, ..., T15.
- The sequence counter along with the decoder on its outputs generate a regular sequence of timing signals that we will refer to as T_0, T_1, etc
- The goal in control unit design is to determine the Boolean function needed for each control input of the registers, bus, and ALU.

Timing and control unit of Basic Computer:



Hardwired control unit design

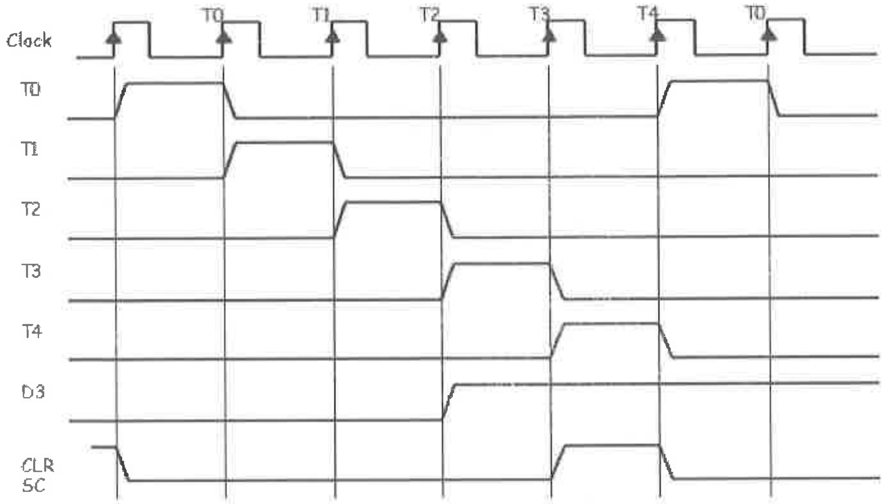
EXAMPLE: D_3T_4: SC_{CLR}

- Generated by 4-bit sequence counter and 4x16 decoder

The SC can be incremented or cleared to generate T_0, T_1, T_2, T_3, T_4, T_0, T_1, . . .

Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active.

D_3T_4: SC ← 0



INSTRUCTION CYCLE:

Every program is executed in the computer by going through a instruction cycle which is in turn 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.

AT T0: Address transfer between PC and AR

AT T0: Since only AR is connected to the address inputs of memory, the address of instruction is transferred from PC to AR.

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 = 010.

2. Transfer the content of the bus to AR by enabling the LD input of AR (AR ←PC).

AT T1: Data transfer between Memory and IR:

T1: The instruction read from memory is placed in IR. At the same time, PC is incremented to the address of the next instruction.

1. Enable 'read input' of the memory.

2. Place the content of memory onto the bus using the bus selection inputs S2S1S0 = 111. (Note that the address lines are always connected to AR, and the next instruction address has been already placed in AR.)

3. Transfer the content of the bus to IR by enabling LD input to IR

(IR ←M[AR]).

1. Increment PC by enabling the INR input of PC (PC ←PC + 1).

AT T2: Decoding:

T2: The operation code in IR is decoded; the indirect bit is transferred to I; the address part of the instruction is transferred to AR

T_0: AR ← PC (S0S1S2=010, T0=1)

T_1: IR ← M[AR], PC ← PC + 1 (S0S1S2=111, T1=1)

T_2: $D_{0\text{-}7}$ ← decoded IR(12-14), AR ← IR(0-11), I ← IR(15)

For every timing cycle, we assume SC ← SC + 1 unless it is stated that SC ← 0.

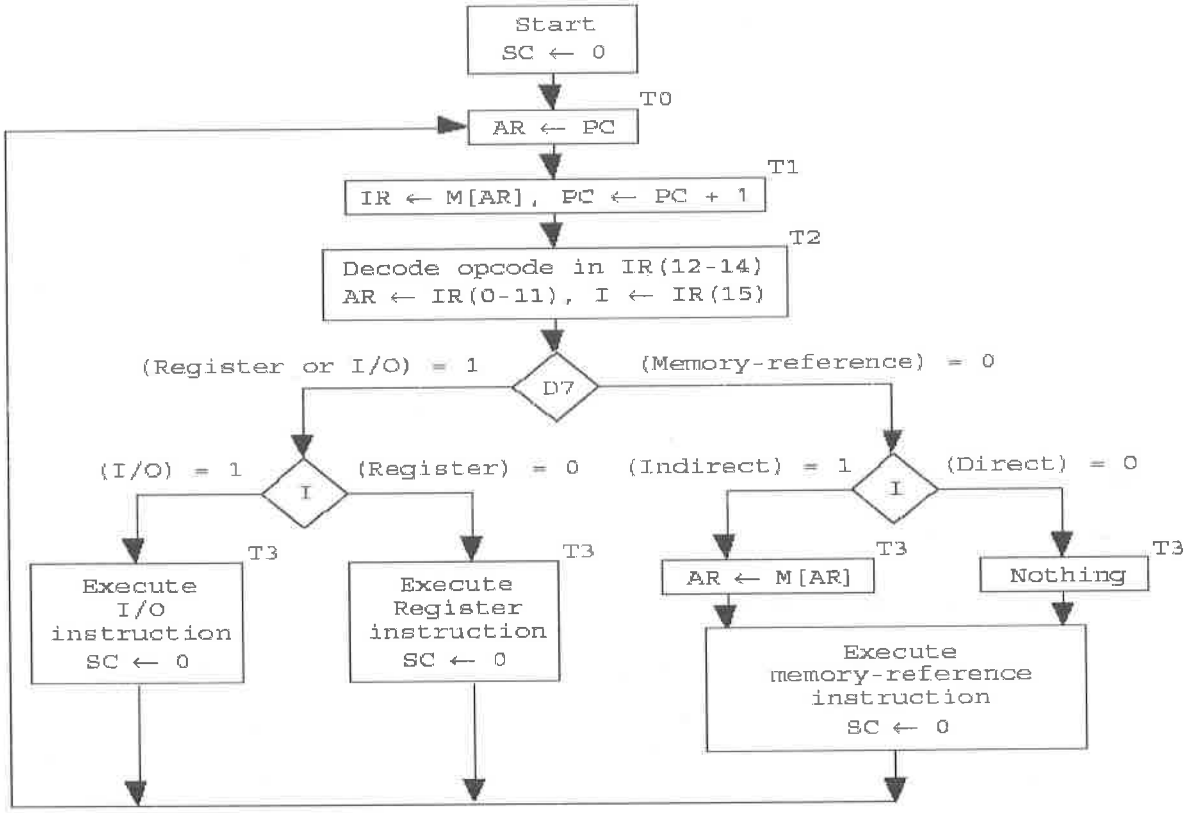


Figure: Flowchart for fetch & decode phases.

D_7' I T_3: AR←M[AR]

D_7' I'T_3: Nothing

D_7I'T_3: Execute a register- reference instr.

D_7IT_3: Execute an input- output instr.

3 b) A machine has 16-bit instruction codes. The Opcode is 4 bits. If there are 3 modes (Immediate, Direct, Indirect) and 4 general-purpose registers, determine the maximum number of Memory-Reference Instructions that can be defined.

Ans:

Instruction set:

An instruction set is the complete collection of instructions that a computer's CPU can understand and execute. It defines:

- What operations the processor can perform (add, move, jump, compare, etc.)
- The format of instructions
- Addressing modes
- Data types

Opcode (Operation Code):

An opcode is the part of the instruction that tells the CPU *what operation to perform*.

Examples:

- ADD → opcode for addition
- MOV → opcode for data transfer

Addressing Modes:

* Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced)

1. Immediate Mode:

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

- No need to specify address in the instruction however, operand itself needs to be specified

- Sometimes, require more bits than the address but fast to acquire an operand

EA: Instruction address e.g. ADD R, #2

2. Direct Address Mode:

Instruction specifies the memory address whichcan be used directly to the physical memory

- Faster than the other memory addressing modes

- Too many bits are needed to specify the address for a large physical memory space

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

3. Indirect Addressing Mode:

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

- When the abbreviated address is used, large physical memory can be addressed with a relatively small number of bits

- Slow to acquire an operand because of an additional memory access

EA = M[IR(address)]

SOLUTION:

Given:

- Instruction length = 16 bits
- Opcode field = 4 bits → $2^4 = 16$possible opcode patterns
- Addressing modes = 3 (Immediate, Direct, Indirect)
- General-purpose registers = 4 → needs $\log_2 4 = 2$bits

Step 1 — Bits to encode mode and register

- Modes: $\lceil \log_2 3 \rceil = 2$bits (since 2 bits can represent up to 4 choices).
- Registers: $\log_2 4 = 2$bits.
  So mode + register = $2 + 2 = 4$bits.

Step 2 — Bits left for the address field

Total bits (16) − opcode (4) − mode+register (4) = $16 - 4 - 4 = 8$bits for the address.

Therefore the address field can specify $2^8 = 256$distinct memory locations.

Step 3 — What is being counted? Two reasonable interpretations

1. **If you count every distinct instruction *encoding* that references memory** (i.e., every combination of opcode value, addressing mode, register, and memory address), then:

   $$\text{Total} = (\text{opcodes}) \times (\text{modes}) \times (\text{registers}) \times (\text{addresses})$$

   A memory-reference instruction encoding is defined by:

   - choice of opcode (16),
   - addressing mode (3),
   - register (4), and
   - memory address (256).

   So total number of distinct memory-reference instruction encodings:

   $$16 \times 3 \times 4 \times 256 = 49{,}152$$

   49,152 distinct memory-reference instruction encodings are possible.

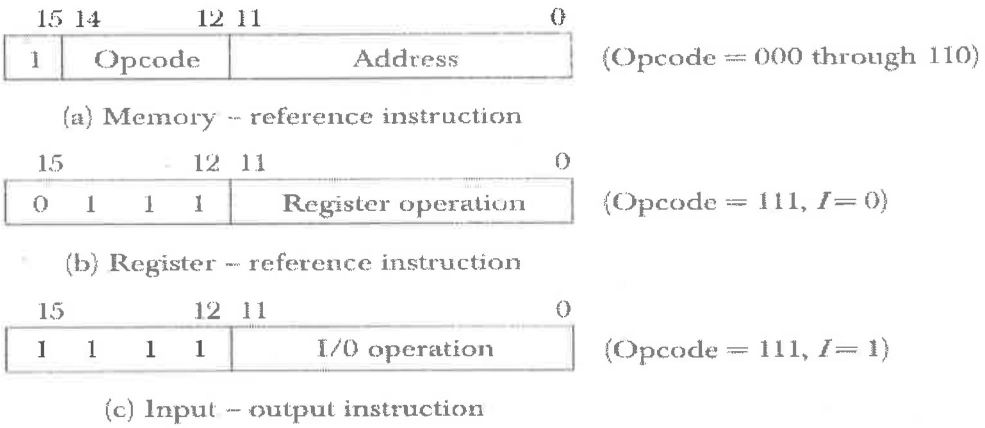
2. **If you only count distinct opcode mnemonics that can be designated as "memory-reference" instructions** (i.e., how many different opcodes you can assign to memory-reference operations, ignoring modes/registers/addresses), then that number is simply the number of opcode patterns: **16**.

4) Explain the concept of Instruction Codes (Memory-Reference, Register-Reference, and Input-Output instructions) and their structure in a basic computer. **14M**

Ans:

Instruction Code: An instruction code is a group of bits that instruct the computer to perform a specific operation (a sequence of micro-operation).

The basic computer has three instruction code formats.



(Opcode = 000 through 110)

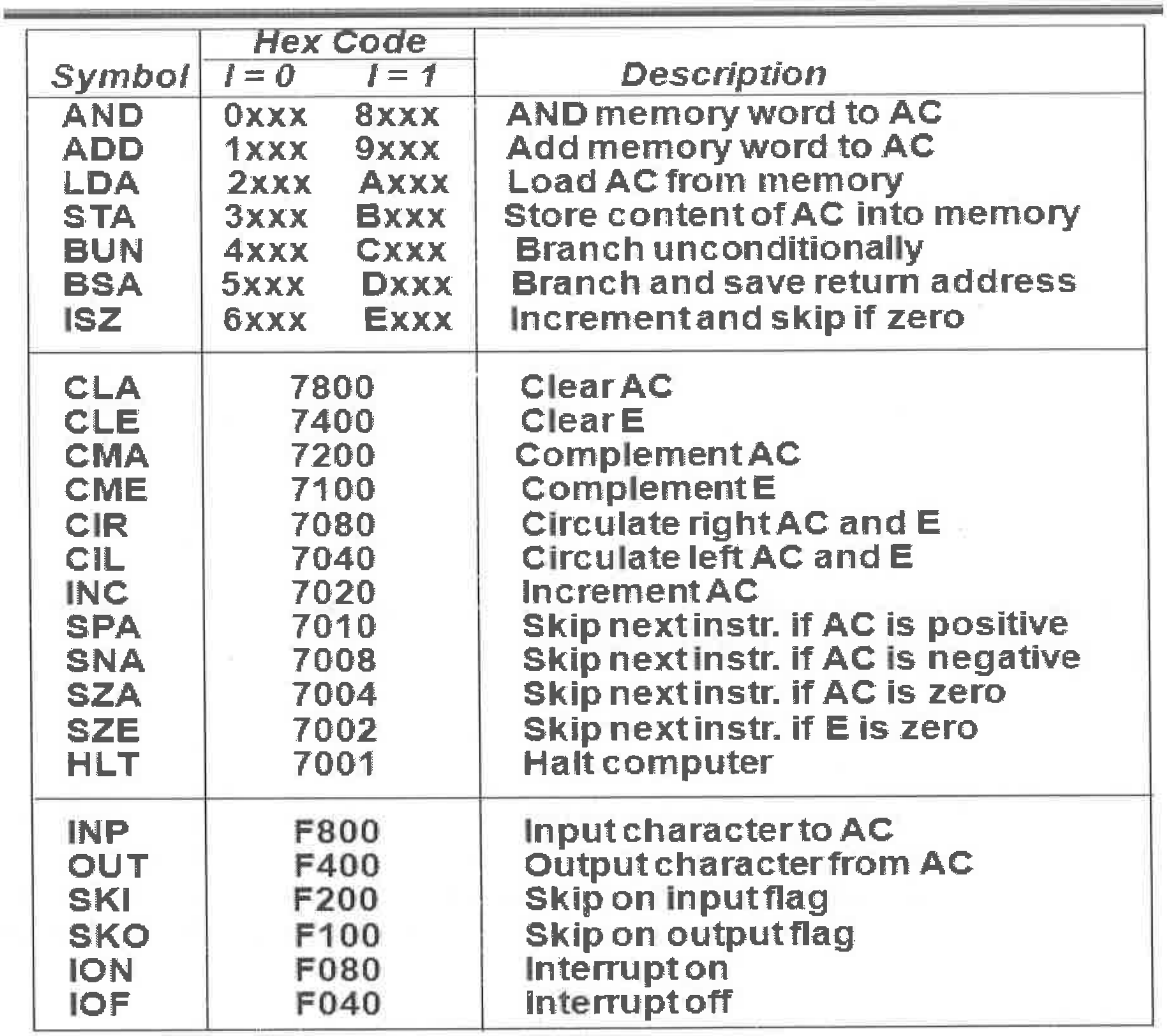
(a) Memory – reference instruction

(Opcode = 111, $I = 0$)

(b) Register – reference instruction

(Opcode = 111, $I = 1$)

(c) Input – output instruction



Symbol	Hex 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 AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

5 a) Explain the influence of the number of addresses on computer program length. Illustrate this by writing the programs for the arithmetic statement X = (A+B) *(C+D) using Three-Address and Zero-Address instruction formats. **7M**

Ans:

The *number of addresses in an instruction format* (e.g., 3-address, 2-address, 1-address, 0-address) affects:

1. Program Length

- **More addresses → fewer instructions needed.**
  - ○ A single instruction can specify more operands.
- **Fewer addresses → more instructions needed.**
  - ○ Operands must be fetched using additional instructions or implicitly from registers/stack.

GIVEN : X=(A+B) ∗(C+D)

Three-Address Instruction Format	**Zero-Address Instruction Format (Stack Machine)**
ADD T1, A, B ; T1 = A + B ADD T2, C, D ; T2 = C + D MUL X, T1, T2 ; X = T1 * T2	PUSH A PUSH B ADD ; stack: (A + B) PUSH C PUSH D ADD ; stack: (C + D) MUL ; multiply the two results POP X ; store result in X
Length • Only **3 instructions** needed. • Each instruction is longer because it has three address fields.	**Length** • **8 instructions**, more than the 3-address version. • Instructions are shorter because they contain no explicit addresses (except PUSH/POP).

5 b) Explain the concept of a Stack Organization in a CPU. Differentiate between a Register Stack and a Memory Stack, and detail the PUSH and POP operations **7M**

Ans:

A. **Stack Organization:**

Stack organization uses a LIFO (Last In, First Out) memory structure controlled by a Stack Pointer (SP) to store temporary data during program execution. Operands are taken from the top of the stack, allowing simpler instructions and faster expression evaluation. It is widely used for arithmetic operations, subroutine calls, interrupts, and storing return addresses. Overall, it simplifies CPU design and improves execution efficiency.

Differences between Register Stack and Memory Stack Organization:

Aspect	Register Stack	Memory Stack
Location	Implemented using a set of high-speed registers inside the CPU.	Implemented in main memory (RAM), specifically in the stack segment.
Stack Pointer (SP)	Small, fixed-size pointer (e.g., 6-bit for 64-word stack). Directly points to registers.	A processor register holds the memory address of the current top of stack (e.g., SP = 4001).
Growth Direction	Typically grows upward (SP increment).	Typically grows downward (SP decrement).
Speed	Very fast (since registers are inside CPU).	Slower compared to register stack (depends on memory access time).
Capacity	Limited (depends on number of registers, e.g., 64 words).	Much larger (depends on allocated memory segment size).
Overflow/Underflow Handling	Hardware can detect FULL/EMPTY conditions using flags (FULL, EMPTY).	Most CPUs don't provide hardware checks; overflow/underflow must be handled via software using **Upper Limit** and **Lower Limit registers**.
Initial Condition	SP = 0, EMPTY = 1, FULL = 0.	SP initialized to the highest address of stack segment (e.g., 4001).

Push Operation	SP ← SP + 1 ; M[SP] ← DR.	SP ← SP - 1 ;
		M[SP] ← DR.

Pop Operation	DR ← M[SP] ;
	SP ← SP - 1.
		DR ← M[SP] ; SP ← SP + 1.

Use Case	Suitable for small, temporary storage (expression evaluation, arithmetic).	Suitable for larger, dynamic storage (function calls, recursion, program execution).

PUSH AND POP OPERATIONS:

PUSH	POP
If the stack is not full (if FULL=0), new item is inserted with a push operation SP ← SP + 1 : Increment stack pointer M[SP] ← DR : Write item on top of the stack if (SP=0) then (FULL ← 1) : Check if stack is full EMPTY ← 0 : Mark the stack not empty	A new item is deleted from stack if the stack is not empty (if EMTY=0) DR ← M[SP] : Read item from the top of stack SP ← SP – 1 : Decrement stack pointer if (SP=0) then (EMTY ← 1): Check if stack if empty FULL ← 0 : Mark the stack full

6 a) With a neat block diagram, explain the General Register Organization of a CPU, focusing on how the common bus system and the decoder select the source and destination registers. **7M**

Ans:

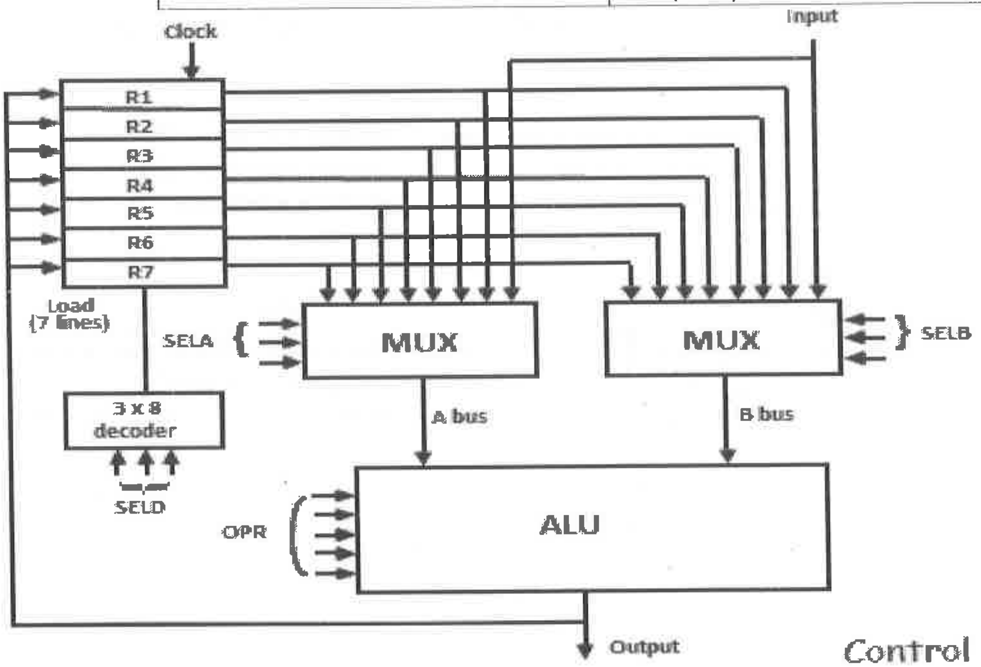
The **General Register Organization** in a CPU uses:

- A **common bus system** to efficiently transfer data among registers.
- **Decoders** to select:
  - **One source register** to put data on the bus.
  - **One destination register** to load data from the bus.
  - **The ALU function** for computation.

This system minimizes hardware while allowing any register to interact with any other register through a uniform mechanism.

Summary of Operation:

Component	**Purpose**
Registers	Store data
Common Bus	Transfers data among registers and ALU
Source Decoder	Selects register to place data onto bus
Destination Decoder	Selects register to receive data from bus
ALU Decoder	Selects the operation (add, AND, shift, etc.)



Control Word

3	3	3	5
SELA	SELB	SELD	OPR

6 b) Explain the Internal Architecture of the 8086 Microprocessor, focusing on the segregation into the Bus Interface Unit (BIU) and the Execution Unit (EU). **7M**

Ans:

The 8086 is mainly divided into mainly two blocks.

1. Execution Unit (EU)
2. Bus interface Unit (BIU)

Dividing the work between these two will speedup the processing

EXECUTION UNIT(EU) : The Execution unit tells the BIU where to

1.Fetch instructions or data from

2. Decodes instructions and

3. Executes instructions

The Execution unit contains:

1) Control circuitry
2) ALU
3) FLAGS
4) General purpose Registers
5) Pointer and Index Registers

BUS INTERFACE UNIT (BIU): The BIU sends out

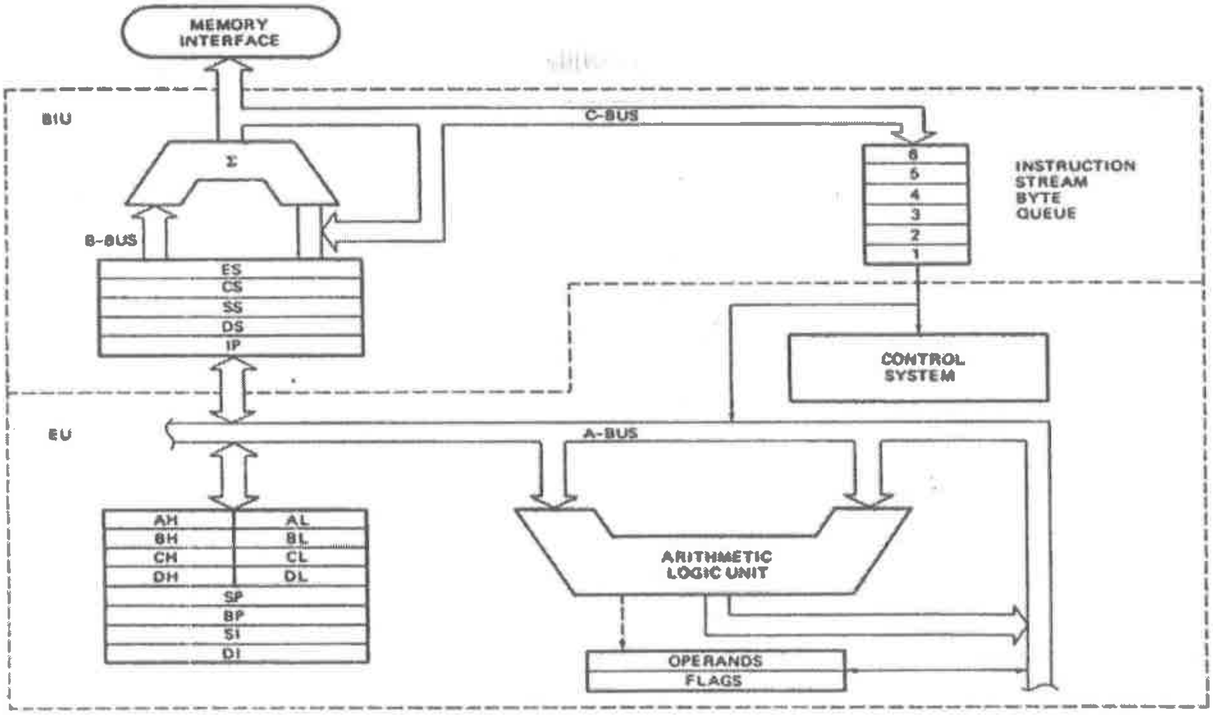
- Addresses
- Fetches instructions from memory
- Read data from ports and memory Or

The BIU handles all transfer of data and addresses on the buses for the Execution Unit.

The Bus interface unit contains

1) Instruction Queue
2) Instruction pointer
3) Segment registers
4) Address Generator

8086 ARCHITECTURE:

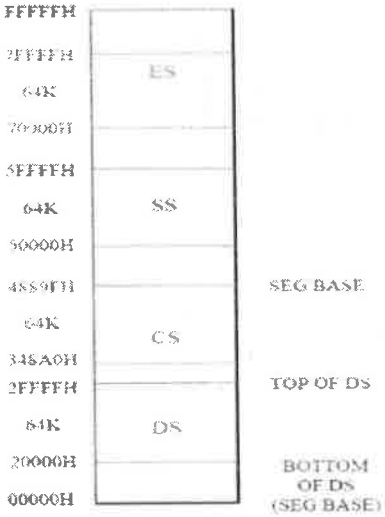


FLAG REGISTERS:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AC	X	PF	X	CY

X = Undefined

MEMORY SEGMENTATION:



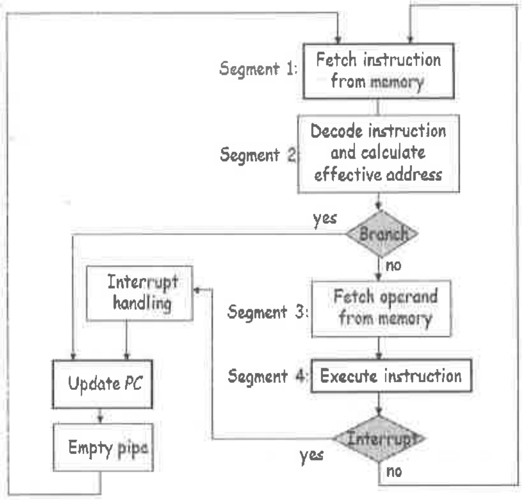
7) Explain the concept of an Instruction Pipeline. Illustrate the speedup achieved using a four-stage pipeline. Discuss the three major types of pipeline conflicts (hazards) that cause the pipeline to stall. 14M

Ans:

Pipeline processing can occur not only in the *data stream* but in the *instruction* as well. In the most general case, the computer needs to process each instruction with the following sequence of steps. There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate.

- Different segments may take different times to operate on the incoming information.
- Some segments are skipped for certain operations.
- Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

Example: Four-segment instruction pipeline:



Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
1	FI	DA	FO	EX									
Instruction: 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

FI: the segment that fetches an instruction
DA: the segment that decodes the instruction and calculate the effective address
FO: the segment that fetches the operand
EX: the segment that executes the instruction

Three major types of pipeline conflicts are:

In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. *Resource conflicts* caused by access to memory by two segments at the same time. (resource or structure hazard)Can be resolved by using separate instruction and data memories

2. *Data dependency conflicts* arise when an instruction dependences between instruction (Data Hazard). An instruction execution depends on the result of a previous instruction, but this result is not yet available.

Example: Result produced by I1 is read by I2.

3. *Branch difficulties* arise from branch and other instructions that change the value of PC. (Control Hazard)

» Default: sequential execution suits pipelining

» Altering control flow (e.g., branching) causes problems

» Introduce control dependencies

8) Describe the three Modes of Transfer for communication between the CPU and I/O devices: Programmed I/O, Interrupt-Initiated I/O, and DMA. State the advantages and disadvantages of each. 14M

Ans:

Modes of transfer :

Data transfer between the central computer and the I/O devices may be handled in a

variety of modes.

The modes of transfer are:

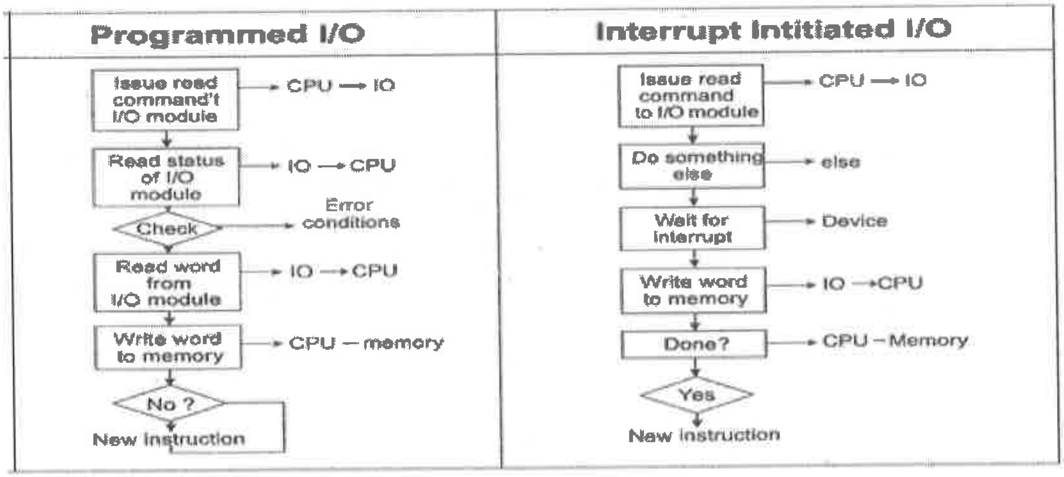
1. Programmed I/O
2. Interrupt-initiated I/O
3. Direct memory access (DMA)

1) Programmed I/O:

- Programmed I/O operations are the result of I/O instructions written in the computer program.
- In this method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer.

2) Interrupt-initiated I/O:

- In the programmed I/O method, the CPU stays in a program 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 interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device.



9 a) With an example, explain the step-by-step procedure of the Booth Multiplication Algorithm for multiplying two signed binary numbers. 7M

Ans: Booth's 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 justshifting, 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 001110 (+14) has a string 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.

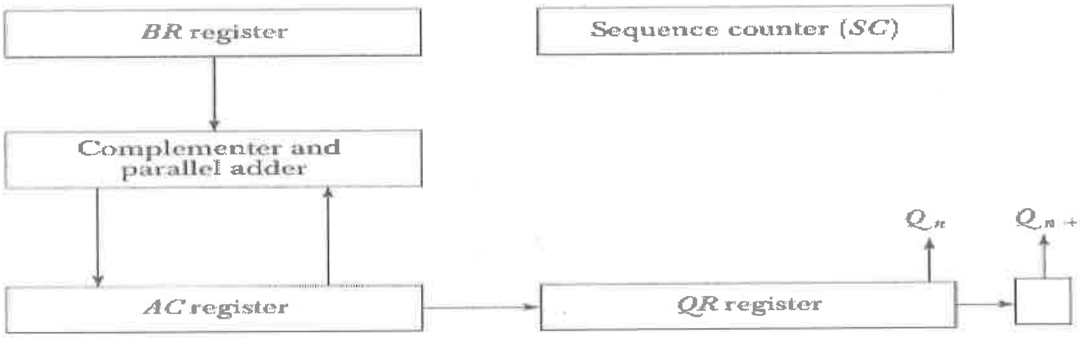


Figure Hardware for Booth algorithm.

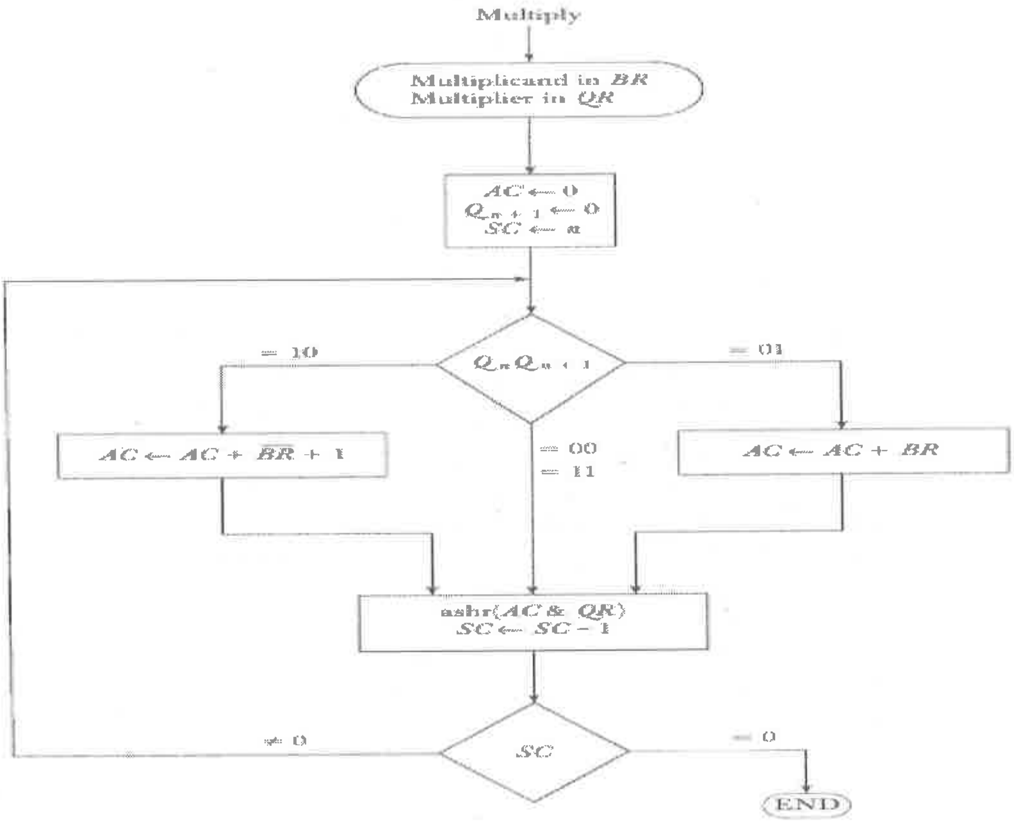


Figure Booth algorithm for multiplication of signed-2's complement numbers.

3) Direct memory access (DMA):

- Direct memory access is an I/O technique used for high-speed data transfer.
- In DMA, the interface transfers data into and out of the memory unit through the memory bus.
- In DMA, the CPU releases the control of the buses to a device called a DMA controller.
- Cycle Stealing:
  DMA Controller acquires control of bus
  Transfers a single byte (or word)
  Releases the bus
- Burst Mode
  DMA Controller acquires control of bus
  Transfers all the data
  Releases the bus

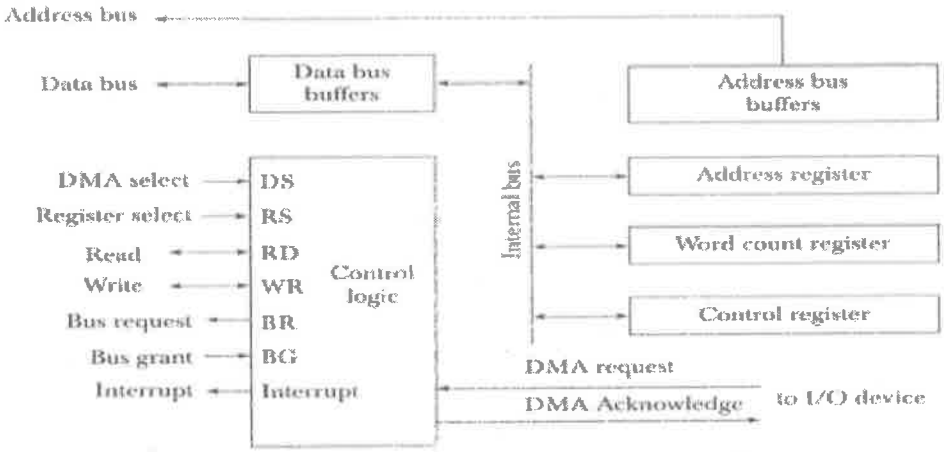
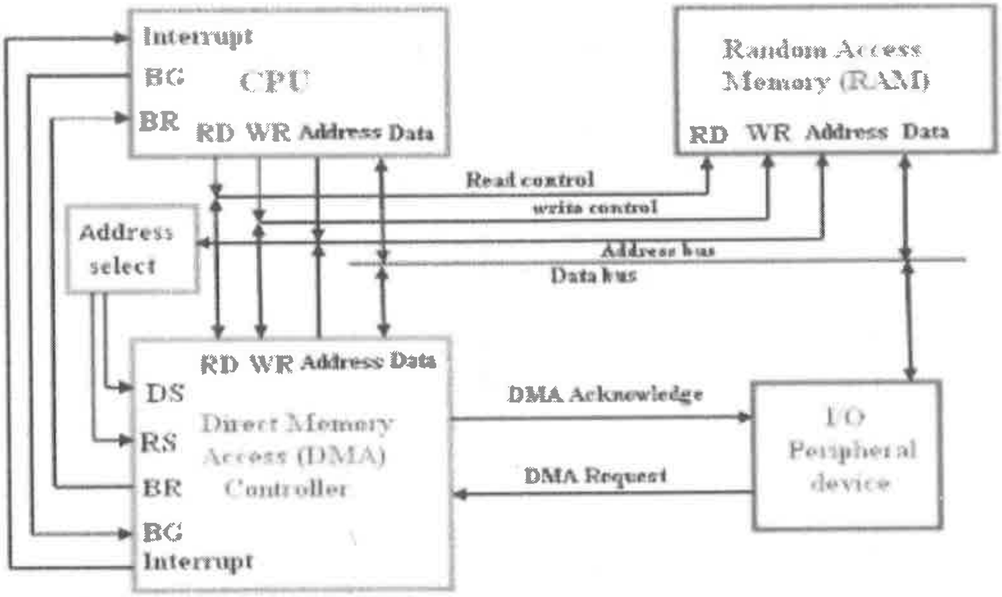


Figure Block diagram of DMA controller.



Advantages:

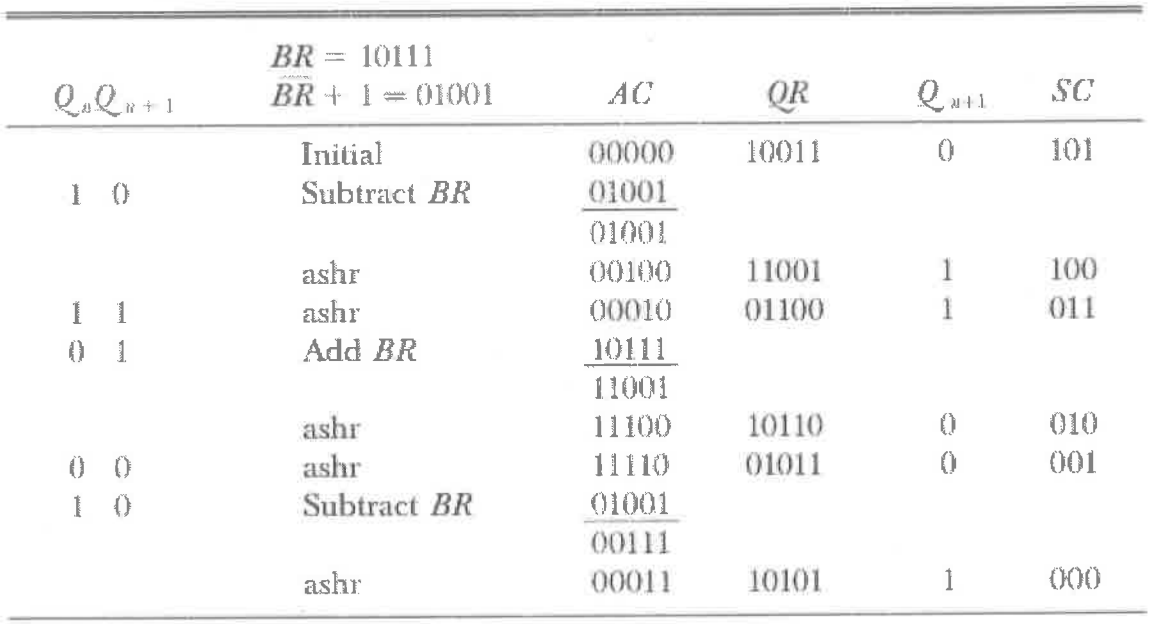
Provides flexibility by allowing different transfer methods based on speed and CPU involvement.

Disadvantages:

Can cause delays such as CPU waiting in programmed I/O or temporary suspension in DMA

ArrayMultiplier:

TABLE Example of Multiplication with Booth Algorithm



Q_nQ_{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

9 b) A computer uses RAM chips of 1024 x 1 capacity. a. How many chips are needed, and how should their address lines be connected to provide a memory capacity of 1024 bytes? b. How many chips are needed to provide a memory capacity of 16K bytes? **7M**

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 when needed

SOLUTION:

Given:

Each RAM chip has a capacity of 1024 x 1

- 1024 locations, 1 bit per location
- Needs 10 address lines (since $2^{10} = 1024$).

a) No of chips needed & No of address Lines for 1024 bytes	**b) No of chips needed & No of address Lines for 16k bytes**
a) Memory capacity required = 1024 bytes **Step 1: Convert bytes into bits** 1 byte = 8 bits 1024 bytes = $1024 \times 8 = 8192$bits Each chip stores **1024 bits** Number of chips required:$\frac{8192 \text{ bits}}{1024 \text{ bits/chip}} = 8$ chips **Step 2: How to connect the address lines** • Each chip needs 10 address lines, because it has 1024 (i.e., 2^{10}) locations. • To make an 8-bit byte, all 8 chips are connected	**b) Memory capacity required = 16K bytes** **Step 1: Convert to bytes** :16K bytes = $16 \times 1024 = 16{,}384$bytes **Step 2: For each 1 KB (1024 bytes), we need 8 chips** (From part a)So for 16 KB: $16 \text{ blocks} \times 8 \text{ chips/block} = 128$ chips **Step 3: Address connections** • Each chip still uses 10 address lines (A0–A9). • To access 16K bytes, total address lines needed:

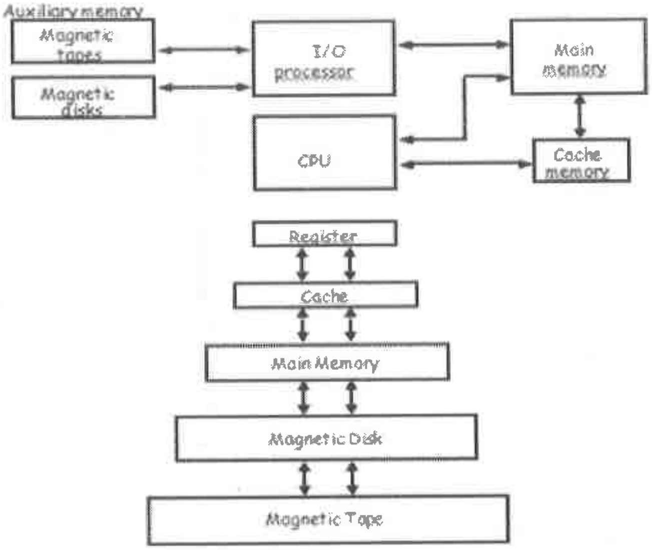
in parallel **Final (a) Answer** • Number of chips required = 8 chips • Address lines: Connect all 10 address lines in parallel to all 8 chips. • The 8 data output lines (1 from each chip) form the 8-bit data bus.	$16K = 2^{14} \Rightarrow$ 14 address lines (A0–A13) • A0–A9 → go to every chip (for internal 1024 locations) • A10–A13 → used for selecting one of the 16 groups of 8 chips (through a decoder). **Final (b) Answer** • Number of chips required = 128 chips • Addressing: ○ A0–A9 to all chips (parallel). ○ A10–A13 to a decoder to select one of the 16 groups of 8 chips.

10 a) Explain the three levels of the Memory Hierarchy in a computer system (Cache, Main Memory, Secondary Storage). Why is this hierarchy essential for system performance? 7M

Ans:

Memory Hierarchy:

Memory is used for storing programs and data that are required to perform a specific task. For CPU to operate at its maximum speed, it required an uninterrupted and high speed access to these memories that contain programs and data. Some of the criteria need to be taken into consideration while deciding which memory is to be used:



- Cost
- Speed
- Memory access time
- Data transfer rate
- Reliability

- **Auxiliary Memory:** The auxiliary memory is at the bottom and is not connected with the CPU directly. However, being slow, it is present in large volume in the system due to its low pricing. This memory is basically used for storing the programs that are not needed in the main memory. This helps in freeing the main memory which can be utilized by other programs that needs main memory. The main function of this memory is to provide parallel searching that can be used for performing a search on an entire word.

- **Main Memory:** The main memory is at the second level of the hierarchy. Due to its direct connection with the CPU, it is also known as central memory. The main memory holds the data and the programs that are needed by the CPU. The main memory mainly consists of RAM, which is available in static and dynamic mode.

- **Cache Memory:** Cache memory is at the top level of the memory hierarchy. This is a high speed memory used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. Cache memory is usually placed between the CPU and the main memory.

❖ Importance of Memory Hierarchy for System Performance:

The memory hierarchy is essential because it balances **speed, cost, and capacity** by placing the fastest but smallest memories (registers, cache) closest to the CPU. This reduces access time and keeps the processor busy instead of waiting for slow main memory or storage. As a result, overall **system performance and efficiency increase significantly.**

10 b) Explain Cache with associative and two way Set-Associative mapping with a line size of 4 bytes? 7M

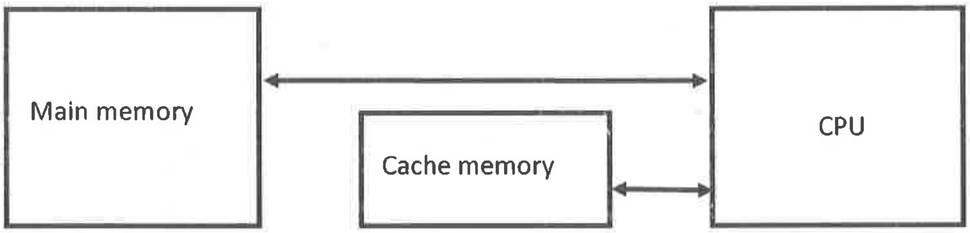
Ans:

. Cache Memory

- The property of Locality of Reference makes the Cache memory systems work.

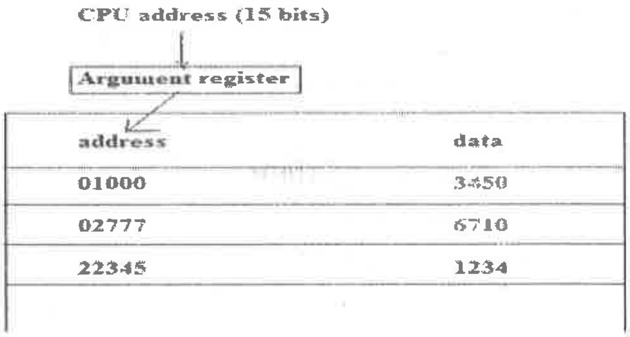
- Cache is a fast small capacity memory that should hold those information

which are most likely to be accessed.



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 any word from main memory
- Any block location in Cache can store any block in memory
- ⇒ Most flexible- Mapping Table is implemented in an associative memory
- ⇒ Very Expensive - Mapping Table Stores both address and the content of the memory word
- 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

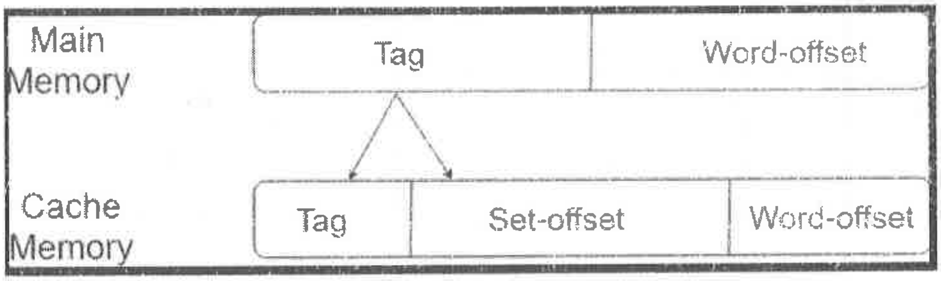


address	data
01000	3450
02777	6710
22345	1234

- 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

Two-way 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



Operation

- CPU generates a memory address (TAG; INDEX)

- Access Cache with INDEX, (Cache word = (tag 0, data 0); (tag 1, data 1))

- Compare TAG and tag 0 and then tag 1

- If tag i = TAG ⇒ Hit, CPU ← data i

- If tag i ≠ TAG ⇒ Miss,

Replace either (tag 0, data 0) or (tag 1, data 1),

Assume (tag 0, data 0) is selected for replacement,

(Why (tag 0, data 0) instead of (tag 1, data 1)?)

M [tag 0, INDEX] ← Cache [INDEX](data 0)

Cache [INDEX](tag 0, data 0) ← (TAG, M [TAG, INDEX]);

CPU ← Cache [INDEX] (data 0)

Prepared by,