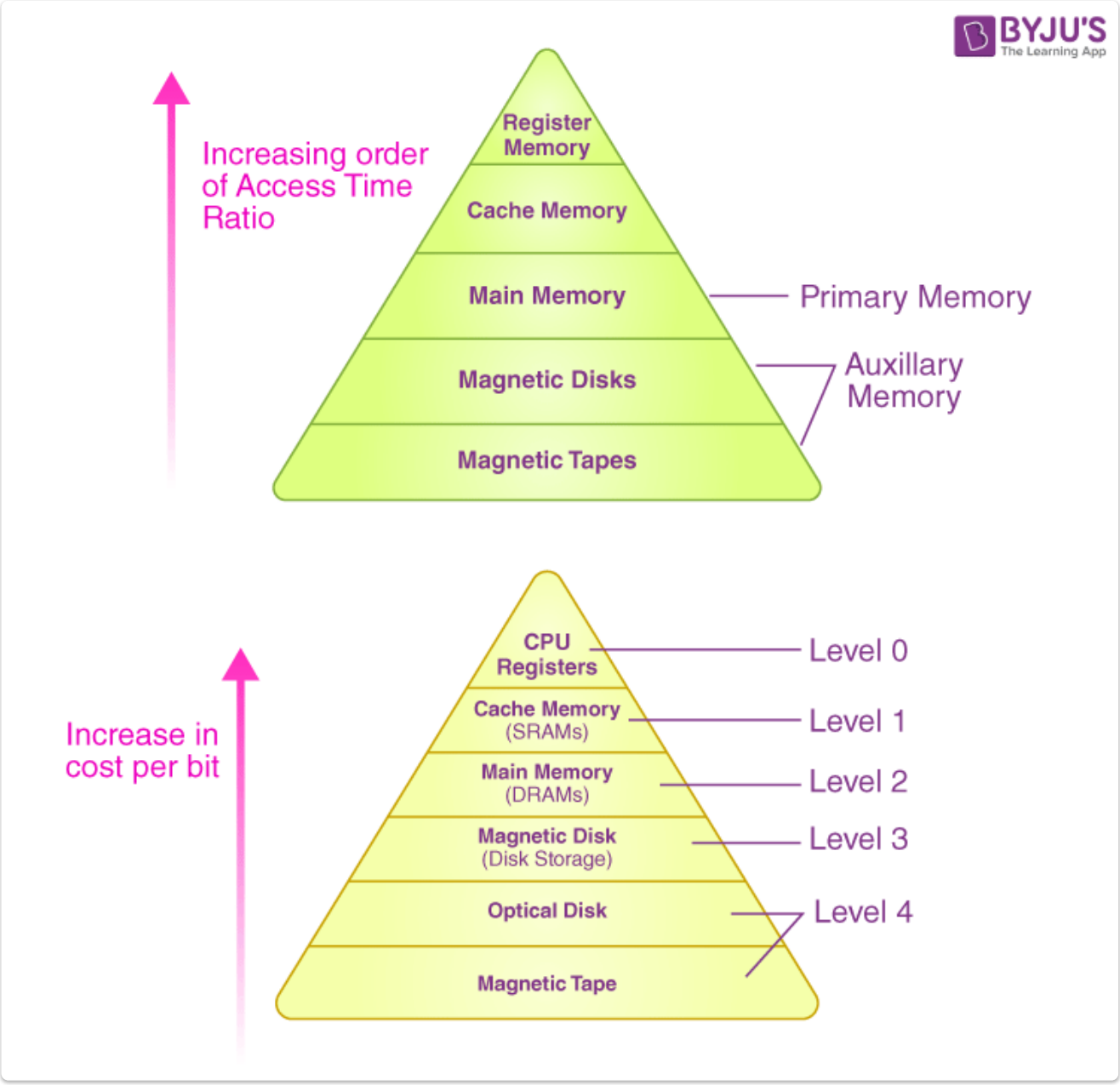
COA Important Question

1.1 Discuss the Memory Hierarchy in computer system with regard to Speed, Size and Cost.

Memory Hierarchy:

The memory hierarchy in a computer system is a structured arrangement of various storage components, each with different speeds, sizes, and roles. This hierarchy aims to optimize the trade-off between speed, size, and cost. It includes auxiliary memory, main memory, and cache memory.



Components of Memory Hierarchy:

1. Auxiliary Memory:

- Slowest component in the hierarchy.
- Access time is approximately 1000 times slower than main memory.
- Used for backup storage.
- Examples include magnetic disks, tapes, drums, bubble memory, and optical disks.
- Accessed through Input/Output channels.
- **Speed:** Slowest in the hierarchy.
- **Size:** Largest storage capacity.
- **Cost:** Economical per unit of storage.
- **Role:** Provides non-volatile, long-term storage. Examples include hard drives, SSDs, and optical storage.

2. **Main Memory:**
    - Central storage unit of the computer system.
    - Communicates directly with the CPU and auxiliary memory devices.
    - Made up of Random Access Memory (RAM) and Read-Only Memory (ROM).
    - RAM includes Dynamic RAM (DRAM), Static RAM (SRAM), and Non-Volatile RAM (NVRAM).
    - ROM is non-volatile and contains the bootstrap loader program for starting the operating system.
    - **Speed:** Faster than auxiliary memory but slower than cache.
    - **Size:** Moderate capacity, smaller than auxiliary memory.
    - **Cost:** More expensive than auxiliary memory.
    - **Role:** Directly interacts with the CPU. Holds active programs and data during execution. Examples include RAM.
3. **Cache Memory:**
    - Used to store frequently accessed program data to speed up CPU access.
    - Smaller and faster than main memory.
    - Accessed before main memory, and data not found in cache results in accessing main memory.
    - Stores blocks of recent data and replaces old data to accommodate new information.
    - Access time ratio between cache memory and main memory is approximately 1 to 7~10.
    - **Speed:** Fastest in the hierarchy.

- **Size:** Smallest storage capacity.
- **Cost:** Most expensive per unit of storage.
- **Role:** Acts as a high-speed buffer between the CPU and main memory. Stores frequently accessed data for quick retrieval. Examples include L1, L2, and L3 caches.

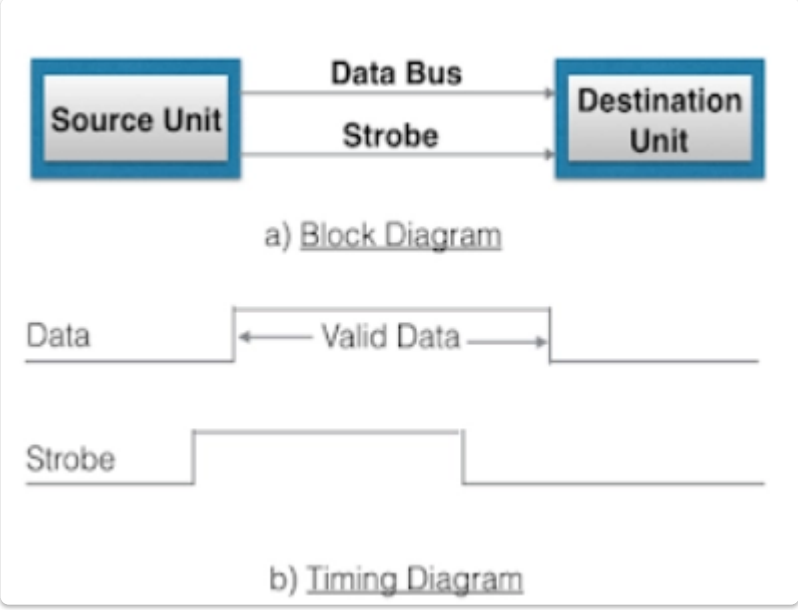
1.2 Explain in detail about strobe control method of asynchronous data transfer.

Asynchronous Data Transfer: Strobe Control

Asynchronous data transfer is a scheme used when the speed of I/O devices doesn't match that of the microprocessor, and the timing characteristics of I/O devices are unpredictable. This method involves the process initiating the device, checking its status, and the CPU waiting until the I/O device is ready to transfer data. Two techniques are used based on signals before data transfer: Strobe Control and Handshaking.

Strobe Signal:

The Strobe Control method uses a single control line to time each transfer. The strobe can be activated by either the source or the destination unit. There are two scenarios for data transfer initiation:



1. **Data Transfer Initiated by Source Unit:**

- The data bus carries binary information from the source to the destination unit.
- The strobe signal, when activated by the source unit, informs the destination unit when a valid data word is available.
- In the timing diagram, the source unit first places the data on the data bus, and both the data bus and strobe signal remain active until the destination unit receives the data.

2. **Data Transfer Initiated by Destination Unit:**
   - In this method, the destination unit activates the strobe pulse, informing the source to provide the data.
   - The source responds by placing the requested binary information on the data bus.
   - The data must be valid and remain on the bus long enough for the destination unit to accept it.
   - Once accepted, the destination unit disables the strobe, and the source unit removes the data from the bus.

Disadvantages of Strobe Signal:
The primary disadvantage of the strobe method is the lack of acknowledgment between the source and destination units. The source unit initiating the transfer has no confirmation that the destination unit has received the data, and vice versa. This uncertainty is addressed by the Handshaking method.

1.3 Explain about arithmetic pipelining.

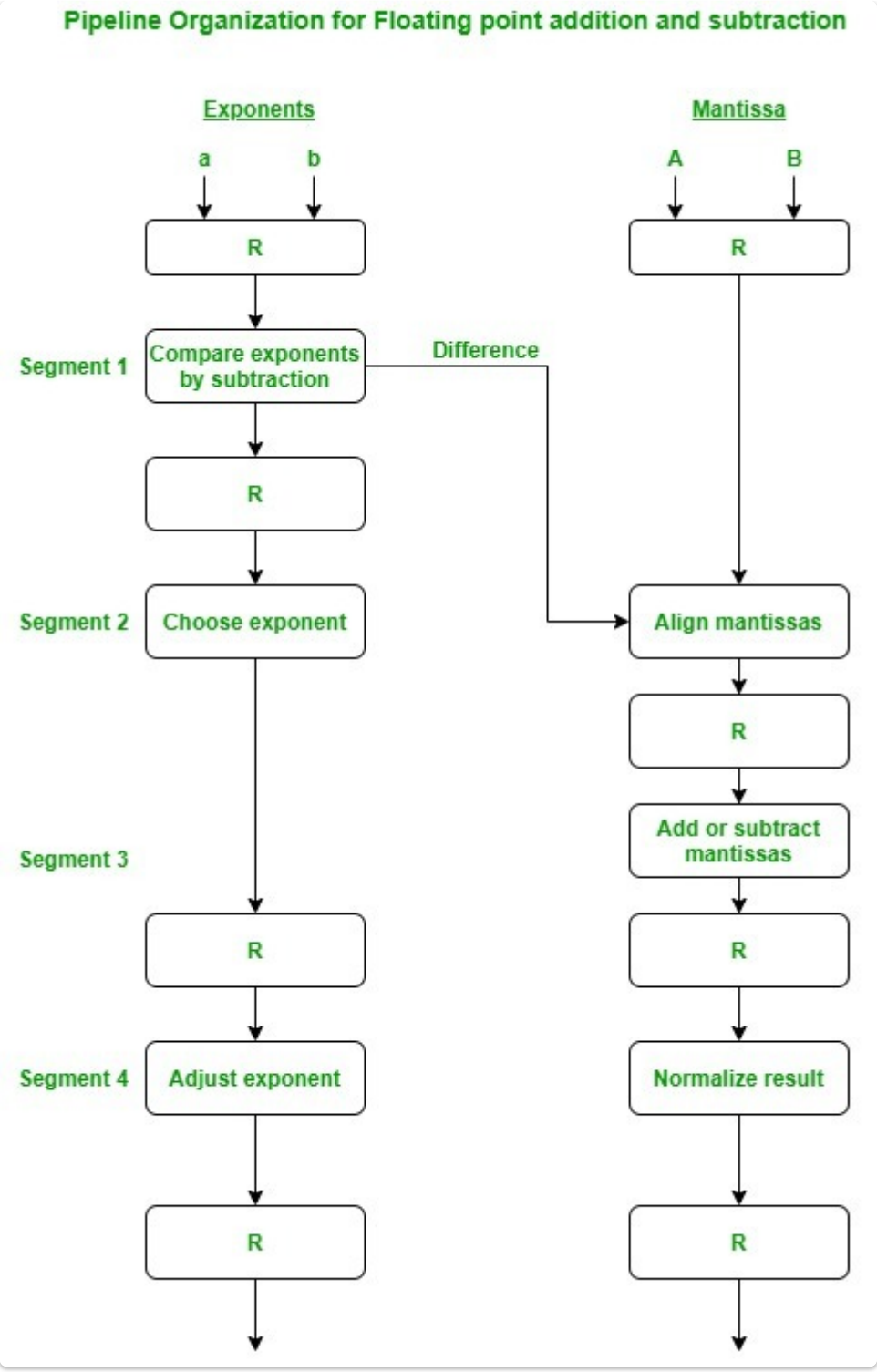
Arithmetic Pipeline for Floating-Point Addition and Subtraction

Arithmetic pipelines are commonly found in high-speed computers and are particularly used for implementing floating-point operations. Let's explore the pipeline unit for floating-point addition and subtraction, involving four segments:

Floating-Point Adder Pipeline:
Floating-point addition pipeline stages:

1. **Compare exponents:**
   - In this step, the exponents of the two floating-point numbers (A and B) are compared. The result of the comparison determines the shift required to align the mantissas properly.

2. **Align mantissas:**
    - After comparing exponents, the mantissas (A and B) are aligned based on the result obtained in the exponent comparison step. This alignment ensures that the addition or subtraction can be performed correctly.
3. **Add mantissas (or subtract):**
    - The aligned mantissas are added or subtracted, depending on whether it's a floating-point addition or subtraction operation. The result is a new mantissa.
4. **Normalize result:**
    - The final step involves normalizing the result. This includes adjusting the exponent and mantissa to obtain a properly normalized floating-point number.



Example Calculation:

Given:

$[X = A \times 10^a = 0.9504 \times 10^3]$

$[Y = B \times 10^b = 0.8200 \times 10^2]$

1. **Compare exponents:**
   - $(a - b = 3 - 2 = 1)$
2. **Align mantissas:**
   - Align based on the result of the exponent comparison:
     - $(X = 0.9504 \times 10^3)$

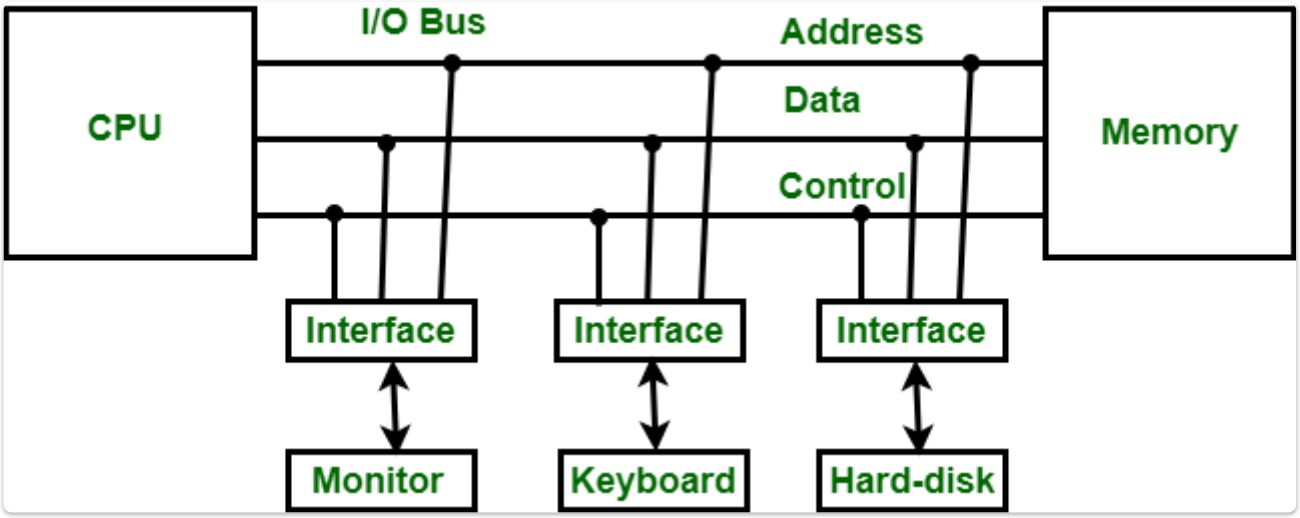
- (Y = 0.08200 \times 10^3)

3. Add mantissas:
   - (Z = 1.0324 \times 10^3)
4. Normalize result:
   - (Z = 0.10324 \times 10^4)

1.4 What is I/O interface and explain it with block diagram.

Input-Output Interface in Computer Systems

The Input-Output (I/O) Interface is a crucial component that facilitates the transfer of information between internal storage and external I/O devices in a computer system. Peripherals connected to a computer require special communication links to interface with the central processing unit (CPU). The purpose of these communication links is to overcome the differences between the central computer and each peripheral device.



Major Differences Addressed by I/O Interface:

1. Conversion of Signals:
   - Peripherals are electromechanical and electromagnetic devices, while the CPU and memory are electronic devices. Signal value conversion may be necessary.
2. Data Transfer Rate:
   - The data transfer rate of peripherals is usually slower than the CPU. A synchronization mechanism may be needed.

3. **Data Codes and Formats:**
    - Data codes and formats in peripherals may differ from the word format in the CPU and memory.
4. **Operating Modes:**
    - The operating modes of peripherals vary, and they must be controlled not to disturb the operation of other connected peripherals.

To address these differences, computer systems include special hardware components known as Interface Units. These units interface between the processor bus and peripheral devices, supervising and synchronizing all input and output transfers.

I/O Bus and Interface Module:

- The I/O Bus consists of data lines, address lines, and control lines.
- A processor communicates with a specific device by placing its address on the address lines.
- Interface Units decode the address, interpret control signals, synchronize data flow, and supervise transfers between peripherals and the processor.
- Each peripheral has its own controller, such as a printer controller for managing paper motion and print timing.

Control Commands in I/O Interface:

- Control Command: Activates the peripheral and informs it about the required action.
- Status Command: Tests various status conditions in the interface and the peripheral.
- Data Output Command: Transfers data from the bus into the interface's registers.
- Data Input Command: Receives data from the peripheral and places it in the buffer register.

I/O Versus Memory Bus:

- Communication with I/O involves interacting with the memory unit. Various methods include separate buses for memory and I/O, a common bus with separate control lines, or a common bus for both memory and I/O with shared control lines.

I/O Processor (IOP):

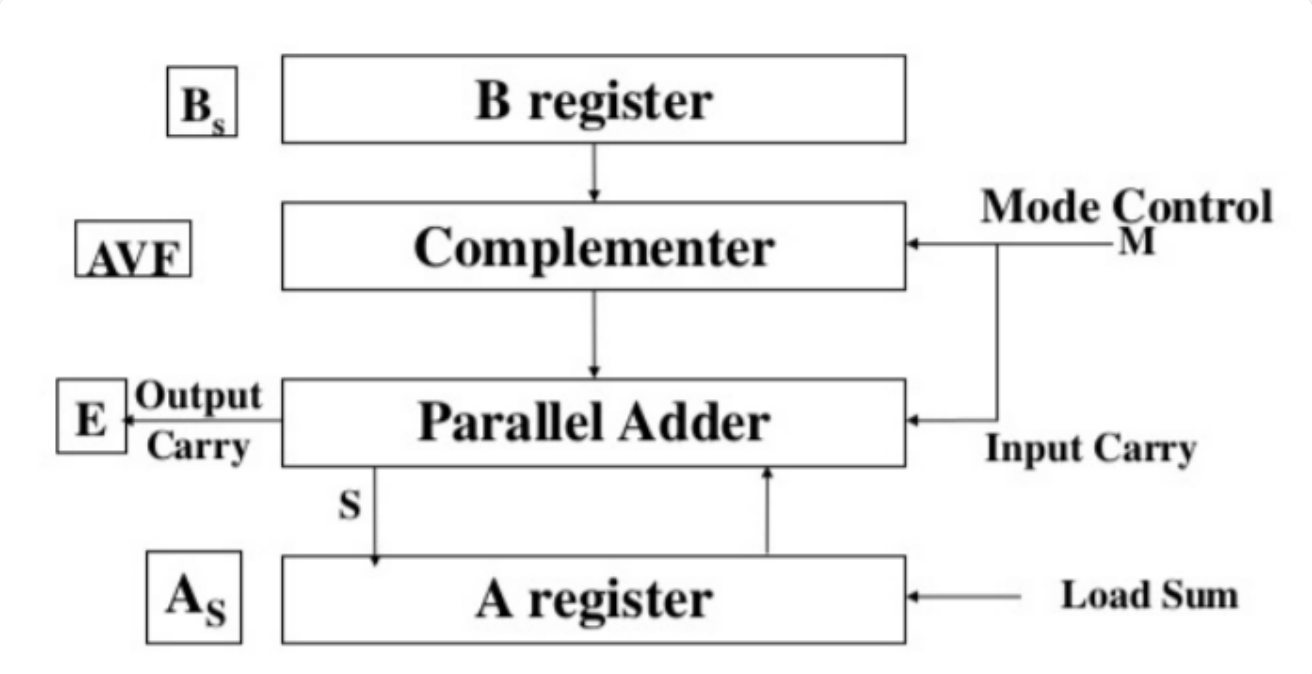
- Some computers utilize an independent I/O processor (IOP) to provide a separate pathway for information transfer between external devices and internal memory.

1.5 Describe the hardware implementation for addition and subtraction in detail.

The hardware implementation for addition and subtraction of signed-magnitude numbers involves several components, including registers, flip-flops, adders, and control logic. Below is a detailed description of the hardware implementation for addition and subtraction:

Block Diagram:

- A block diagram includes A and B registers, As and Bs flip-flops, an adder, a complementer, overflow detection logic, and various control signals.



Components:

1. Registers:
    - A Register: Holds the magnitude of operand A.
    - B Register: Holds the magnitude of operand B.

- **E Register:** Stores the result of the comparison of signs (exclusive-OR gate output) and is used in the subtraction process.

2. **Flip-Flops:**
   - **As (Sign of A) Flip-Flop:** Indicates the sign of operand A.
   - **Bs (Sign of B) Flip-Flop:** Indicates the sign of operand B.
   - **AVF (Add-Overflow) Flip-Flop:** Holds the overflow bit when A and B are added.
3. **Adder:**
   - **Parallel Adder:** Performs the addition of magnitudes.
4. **Complementer:**
   - **Complementer for B:** Generates the 2's complement of operand B for subtraction.
5. **Control Logic:**
   - **Mode Control:** Determines the operation mode (addition or subtraction).
   - **Overflow Detection Logic:** Checks for overflow conditions during addition.

Operation:

Addition:

1. The signs of A and B are compared using an exclusive-OR gate.
2. If the output is 0, the signs are identical, and the magnitudes are added using the parallel adder. The result is stored in the A register.
3. The overflow bit is determined and stored in the AVF flip-flop.

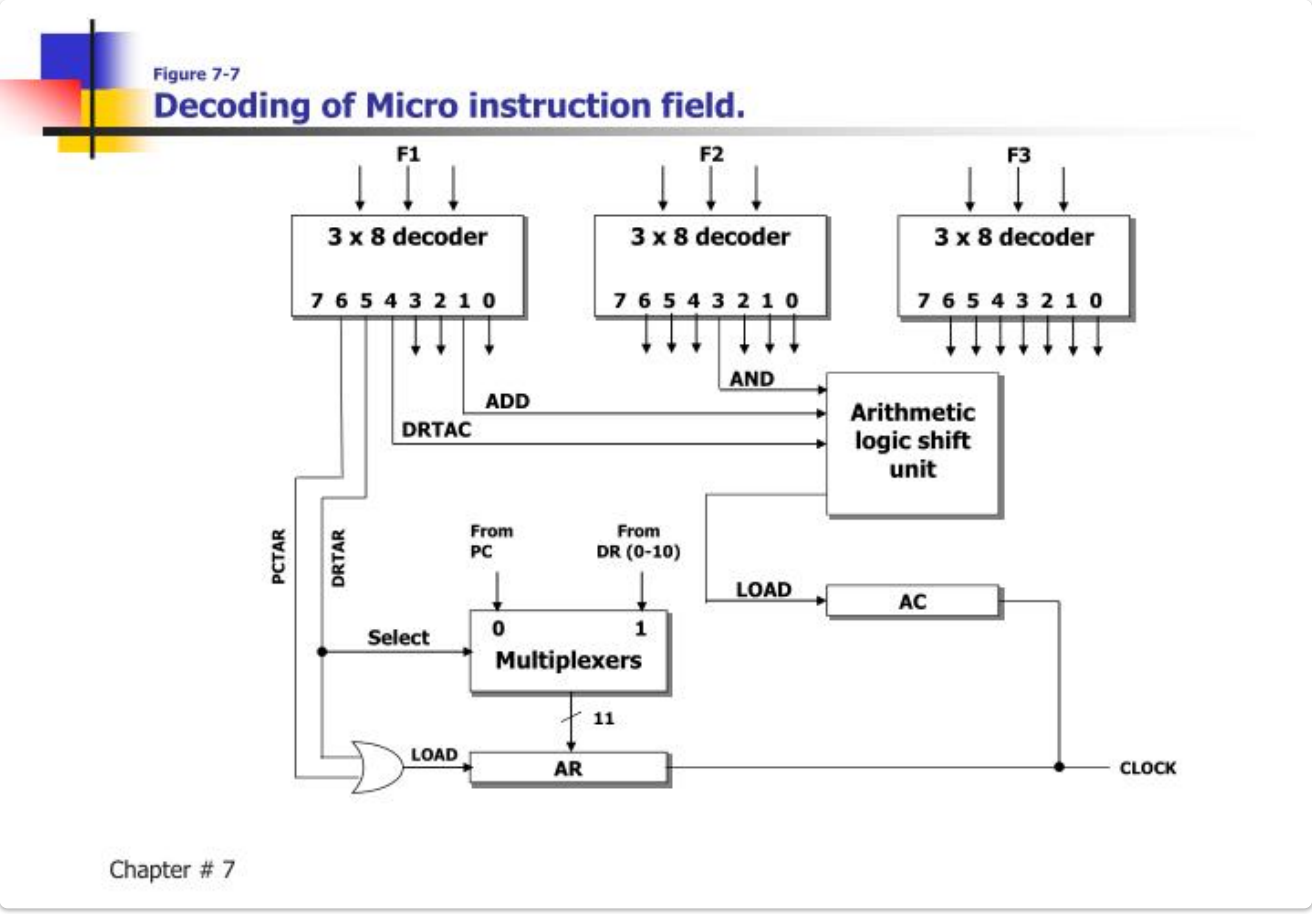
Subtraction:

1. The signs of A and B are compared using an exclusive-OR gate.
2. If the output is 1, the signs are different, and the magnitudes are added using the parallel adder. The result is stored in the A register.
3. The overflow bit is determined and stored in the AVF flip-flop.
4. If the output is 0, the signs are identical, and subtraction is performed by adding A to the 2's complement of B.
5. The E register is used to check whether A is greater than or equal to B. If A < B, the value in A is complemented.
6. The final result is found in the A register, and the sign is determined by the As flip-flop. The AVF flip-flop provides an overflow indication.

1.6 Draw and explain about micro program control unit.

Design of Control Unit

The design of a control unit is a critical aspect of a computer's architecture, and it involves decoding microinstructions to initiate distinct microoperations. Here's an overview of the design considerations and connections for a control unit:



1. Microinstruction Fields:
    - Microinstructions typically consist of multiple fields specifying various aspects of the operation.
    - For example, a microinstruction might have fields like F1, F2, and F3.
2. Decoding:
    - Each field of the microinstruction output from the control memory needs to be decoded to provide specific microoperations.
    - Decoders are used to convert binary values into distinct outputs based on the microinstruction fields.
3. Connection to Processor Unit:

- The outputs of the decoders are connected to the appropriate inputs in the processor unit.
- These connections determine the actions to be performed based on the decoded microinstruction.

4. **Multiple Decoders:**
   - The figure shows three decoders, each responsible for decoding a specific field of the microinstruction.
   - For example, a 3x8 decoder might be used for a 3-bit field to provide eight distinct outputs.
5. **Microoperation Initiation:**
   - Each decoder output is connected to the corresponding circuitry to initiate the specified microoperation.
   - Connections must be made to ensure that the correct action occurs based on the decoded values.
6. **Example Connections:**
   - As an example, when F1 equals 101 (binary 5), there is a transfer from DR to AR.
   - Similarly, when F1 equals 110 (binary 6), there is a transfer from PC to AR (PCTAR).
   - Outputs 5 and 6 of decoder F1 are connected to the load input of AR, determining whether information from DR or PC is transferred to AR.
7. **Multiplexer Usage:**
   - Multiplexers can be employed to select information sources based on specific conditions.
   - For instance, the transfer into AR might occur with a clock transition only when the appropriate decoder output is active.
8. **Arithmetic Logic Shift Unit:**
   - For units like the arithmetic logic shift unit, control signals are generated based on the outputs of logical gates.
   - These signals may come from the outputs of AND, ADD, and DRTAC, indicating the specific operations to be performed.

2.1 Discuss about addressing modes?

Addressing Modes

Addressing modes play a crucial role in determining how operands are chosen during program execution. These modes provide programming versatility and help reduce the number of bits in the addressing field of instructions. Here are various addressing modes and their descriptions:

Implied Mode:

- Operands are implicitly specified in the instruction definition.
- For example, "complement accumulator" is an implied-mode instruction, where the operand in the accumulator register is implied.

Immediate Mode:

- The operand is specified in the instruction itself.
- Useful for initializing registers to a constant value.

Register Mode:

- Operands are in registers within the CPU.
- A particular register is selected from a register field in the instruction.

Register Indirect Mode:

- The instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- The selected register contains the address of the operand.

Auto-Increment or Auto-Decrement Mode:

- Similar to register indirect mode, but the register is incremented or decremented after (or before) its value is used to access memory.

Direct Address Mode:

- The effective address is equal to the address part of the instruction.
- The operand resides in memory, and its address is directly given by the instruction.

Indirect Address Mode:

- The address field gives the address where the effective address is stored in memory.

- The instruction fetches the address, then accesses memory again to read the effective address.

Relative Address Mode:

- The content of the program counter is added to the address part of the instruction to obtain the effective address.

Indexed Addressing Mode:

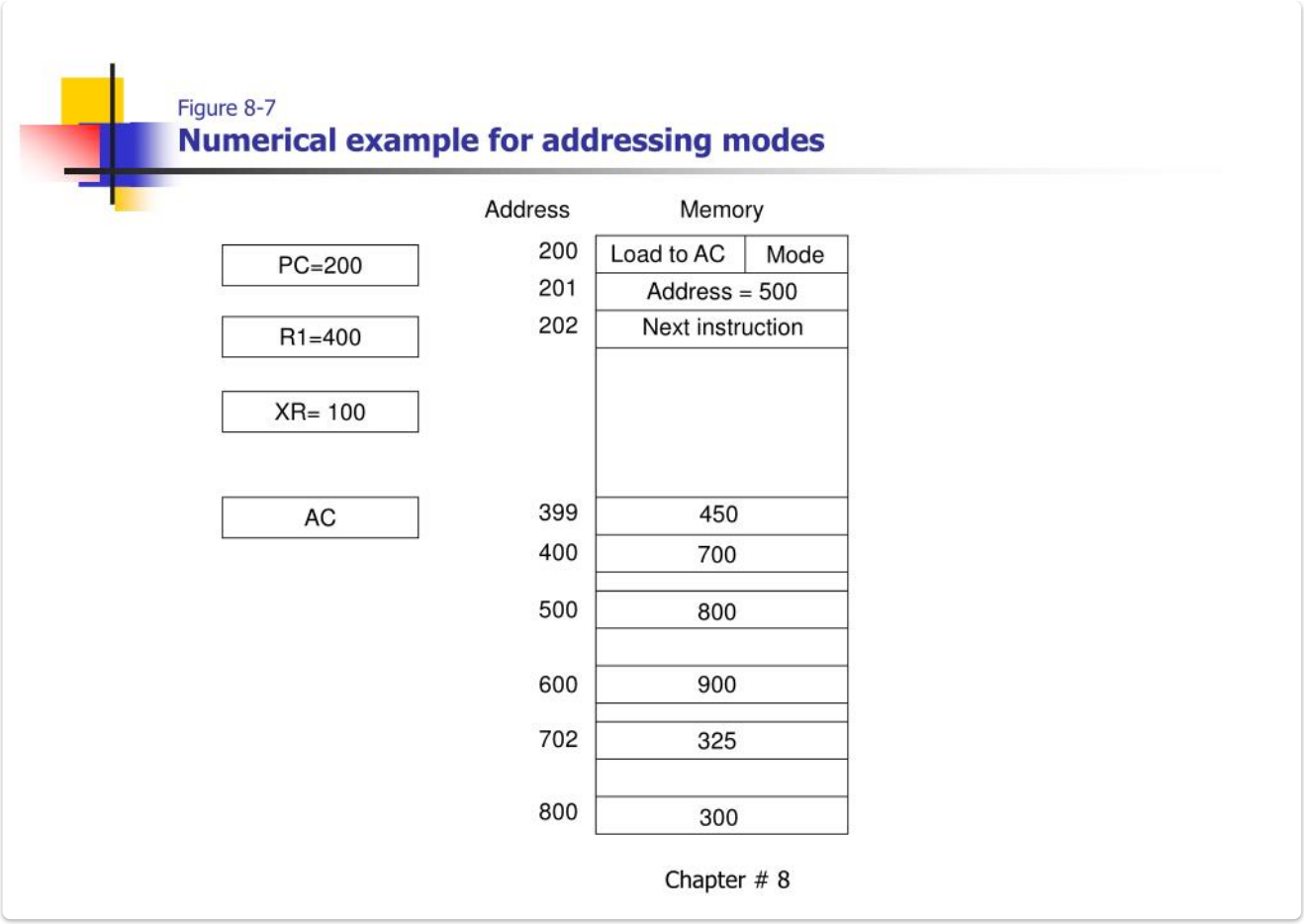
- The content of an index register is added to the address part of the instruction to obtain the effective address.

Base Register Addressing Mode:

- Similar to indexed addressing mode, but a base register is used instead of an index register.

Numerical Example:

- A practical example illustrating the effect of addressing modes on a "load to AC" instruction with an address field equal to 500.
- Direct, immediate, indirect, relative, index, register, register indirect, auto-increment, and auto-decrement modes are considered.
- The table lists the values of effective addresses and operands loaded into AC for each addressing mode.



2.2 Explain the Memory-Reference instructions with examples.

Register-Reference Instructions

Register-reference instructions are a category of instructions recognized by the control unit when D7 = 1 and I = 0. These instructions utilize bits 0 through 11 of the instruction code to specify one of the 12 instructions, with these 12 bits available in the instruction register (IR) (0-11). The control functions and microoperations for these instructions are outlined in Table 5-3.

Execution Timing:

- Register-reference instructions are executed with the clock transition associated with timing variable T3.
- Control functions are determined by the Boolean relation D7I'T3, denoted by the symbol r.
- Control functions are represented as rBi, where Bi is bit i of IR.

Execution of register reference instruction

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

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

Example (CLA Instruction):

- CLA has the hexadecimal code 7800, equivalent to the binary 0111 1000 0000 0000.
- The first bit (I') is zero, and the operation code is recognized from decoder output D7.
- Bit 11 in IR is 1 (B11 = 1), and the control function for this instruction is D7I'T3 * B11 = rB11.
- Execution is completed at time T3, and the sequence counter SC is cleared to 0 to fetch the next instruction with timing signal T0.

Microoperations:

1. The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers.
2. The next four instructions cause a skip of the next instruction in the sequence when a specific condition is satisfied, achieved by incrementing the program counter (PC) once again.
3. Condition control statements are part of the control conditions.
4. AC is considered positive when the sign bit in AC(15) = 0, negative when AC(15) = 1, and zero when AC = 0.
5. The HLT instruction clears a start-stop flip-flop (S) and stops the sequence counter from counting.

2.3 With an example, explain Booth Multiplication algorithm.

Booth's Algorithm for Binary Multiplication

Booth's algorithm provides a systematic procedure for multiplying binary integers in signed 2's complement representation. The algorithm is based on the observation that strings of 0's in the multiplier only require shifting, while a string of 1's in the multiplier from bit weight 2k to weight 2m can be treated as (2^{k+1} - 2^m).

Procedure:

1. **Initialization:**
   - Set the product (P) to zero.
   - Create a working copy of the multiplier (M).
   - Create a copy of the multiplicand (Q) with an extra bit on the left for sign extension.
2. **Repeat for the number of bits in the multiplier (n):**
   a. Examine the last two bits of M and Q.
   b. If the last two bits are "00" or "11," perform a right shift on the product and the working copy of the multiplier.
   c. If the last two bits are "01," perform (P = P - Q), then right shift the product and the working copy of the multiplier.
   d. If the last two bits are "10," perform (P = P + Q), then right shift the product and the working copy of the multiplier.
3. **Continue the iteration until n iterations are completed.**
4. **Result:**
   - The final product is in the product register.

Example:

Let's consider the multiplication of 6 (110) and -3 (1011) using Booth's algorithm.

1. Initialization:
   - Product (P) = 0
   - Multiplier (M) = 1011 (-3)
   - Multiplicand (Q) = 0110 (6)
2. Iterations:
   - (P = P - Q) (because 11 is "01"), right shift P and M.

- (P = P + Q) (because 10 is "10"), right shift P and M.
- Right shift P and M.
- Right shift P and M.

3. Result:
    - The final product is 10010 (-18).

2.4 Explain various Data Manipulation instructions with examples

Data Manipulation Instructions

1. Arithmetic Instructions

- Data manipulation instructions perform operations on data and provide computational capabilities for the computer.
- The data manipulation instructions in a typical computer are usually divided into three basic types:

1. Arithmetic Instructions

- The four basic arithmetic operations are addition, subtraction, multiplication, and division.
- Most computers provide instructions for all four operations.
- Some small computers have only addition and possibly subtraction instructions. Multiplication and division must then be generated by means of software subroutines.
- The increment instruction adds 1 to the value stored in a register or memory word.
- A number with all 1's, when incremented, produces a number with all 0's.
- The decrement instruction subtracts 1 from a value stored in a register or memory word.
- A number with all 0's, when decremented, produces a number with all 1's.
- The add, subtract, multiply, and divide instructions may use different types of data.
- The data type assumed to be in the processor register during the execution of these arithmetic operations is defined by an operation code.

- An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data, single-precision, or double-precision data.
- Examples of mnemonics for three add instructions that specify different data types are shown below:
  - ADDI: Add two binary integer numbers
  - ADDF: Add two floating-point numbers
  - ADDD: Add two decimal numbers in BCD
- Special instructions like "add carry" and "subtract with borrow" deal with the carry from a previous operation.
- The negate instruction forms the 2's complement number, effectively reversing the sign of an integer when represented in signed-2's complement form.

Name	Mnemonic
INCREMENT	INC
DECREMENT	DEC
ADD	ADD
SUBTRACT	SUB
MULTIPLY	MUL
DIVIDE	DIV
ADD WITH CARRY	ADDC
SUBTRACT WITH BORROW	SUBB
NEGATIVE	NEG

Example:

- Addition (ADD):

```
MOV AX, 5
ADD AX, 3  ; AX = AX + 3
```

- Subtraction (SUB):

```
MOV AX, 8
SUB AX, 2  ; AX = AX - 2
```

2. Logical and Bit Manipulation Instructions

- Logical instructions perform binary operations on strings of bits stored in registers.

- They are useful for manipulating individual bits or groups that represent binary-coded information.
- Typical logical and bit manipulation instructions include:
  - Clear
  - Complement
  - AND, OR, XOR
  - Bit manipulation operations: clear, set, complement
- Other bit manipulation instructions perform operations on individual bits, such as clearing, setting, or complementing.

Name	Mnemonic
CLEAR	CLR
COMPLEMENT	COM
AND	AND
OR	OR
EXCLUSIVE OR	XOR
CLEAR CARRY	CLRC
SET CARRY	SETC
COMPLEMENT CARRY	COMC
ENABLE INTERRUPT	EI
DISABLE INTERRUPT	DI

Example:

- AND Operation:

```
MOV AX, 1010b
AND AX, 1100b ; AX = AX AND 1100b
```

- OR Operation:

```
MOV AX, 1010b
OR AX, 1100b  ; AX = AX OR 1100b
```

3. Shift Instructions

- Shifts are operations in which the bits of a word are moved to the left or right.
- Shift instructions may specify logical shifts, arithmetic shifts, or rotate-type operations.

- Types of shift instructions include logical shift left/right, arithmetic shift left/right, rotate left/right, and rotate through carry.
- Shifts may be used for various purposes, including moving bits to designated positions, preserving sign bits in arithmetic shifts, and performing circular shifts.
- A list of typical arithmetic instructions is given in Table 8-7.
- The logical shift inserts 0 into the end bit position.
- Arithmetic shifts usually conform to the rules for signed-2's complement numbers.
- Rotate instructions produce a circular shift, circulating bits shifted out back into the opposite end.
- Rotate through carry treats a carry bit as an extension of the register being rotated.

TABLE 8-9 Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Example:

- Left Shift (SHL or SAL):

```
MOV AX, 00001010b
SHL AX, 2 ; AX = AX << 2
```

- Right Shift (SHR):

```
MOV AX, 00001010b
SHR AX, 1 ; AX = AX >> 1
```

2.5 Perform the arithmetic operation (+42)+(-13) and (-42)-(-13) in binary using signed 2's

complement representation for negative numbers.

To perform arithmetic operations in binary using signed 2's complement representation for negative numbers, we can follow these steps:

1. **Convert numbers to binary:**
   Convert both numbers to their binary representations.
2. **Perform the addition and subtraction:**
   Perform the binary addition for (+42) + (-13) and binary subtraction for (-42) - (-13).

I apologize for the mistake in my previous responses. Let's reevaluate the addition (+42) + (-13) using the correct calculations:

Addition: (+42) + (-13)

1. Convert numbers to 8-bit binary:

```
42 in binary: 00101010
-13 in 2's complement: 11110011
```

2. Perform binary addition:

```
    00101010   (+42)
+   11110011   (-13 in 2's complement)
-----------------
    00011101   (Result in binary)
```

Subtraction: (-42) - (-13)

1. Convert numbers to 8-bit binary:

```
-42 in 2's complement: 11010110
-13 in 2's complement (since subtraction is equivalent to adding
the 2's complement): 11110011
```

2. Perform binary addition:

```
    11010110   (-42 in 2's complement)
+   11110011   (-13 in 2's complement)
-----------------
   110010101   (Result in binary)
```

So, in summary:

- (+42) + (-13) in binary is 10011101 (in 2's complement), which is 29 in decimal.
- (-42) - (-13) in binary is 01000001 (in 2's complement), which is -27 in decimal.

2.6 Write the major characteristics of RISC processors.

Reduced Instruction Set Architecture (RISC) is a type of computer architecture design that emphasizes a simple and minimalistic set of instructions for better performance. The main idea is to make hardware simpler and more efficient by using a streamlined set of instructions that can be executed in a single clock cycle.

Characteristics of RISC:

1. **Simpler Instructions:**
   - RISC architectures use a reduced and simpler set of instructions compared to Complex Instruction Set Computing (CISC) architectures. This simplicity makes instruction decoding straightforward.
2. **Single Word Instructions:**
   - Instructions in RISC architectures are typically designed to fit within a single word. This ensures that each instruction can be fetched and executed in a single clock cycle, leading to faster performance.
3. **Single Clock Cycle Execution:**
   - RISC architectures aim to execute each instruction in a single clock cycle. This results in faster instruction throughput and improved performance.
4. **More General-Purpose Registers:**
   - RISC architectures typically include a larger number of general-purpose registers. This reduces the need for memory access, as more data can be

stored in registers, enhancing overall speed and efficiency.

5. **Simple Addressing Modes:**
    - RISC architectures use simple and efficient addressing modes. This simplicity contributes to faster instruction execution and reduces the complexity of the instruction set.
6. **Less Data Types:**
    - RISC architectures often support a reduced number of data types. This design choice simplifies the hardware and contributes to the efficiency of the instruction set.
7. **Pipeline Architecture:**
    - RISC architectures often employ pipelining, a technique where the processor is divided into stages, and different stages of instruction execution are overlapped. This helps in improving throughput and achieving better performance.

Advantages of RISC:

- Simplicity
- Faster Execution
- Efficient Register Usage
- Compiler-Friendly

Disadvantages of RISC:

- Code Size
- Limited Instructions
- Compatibility
