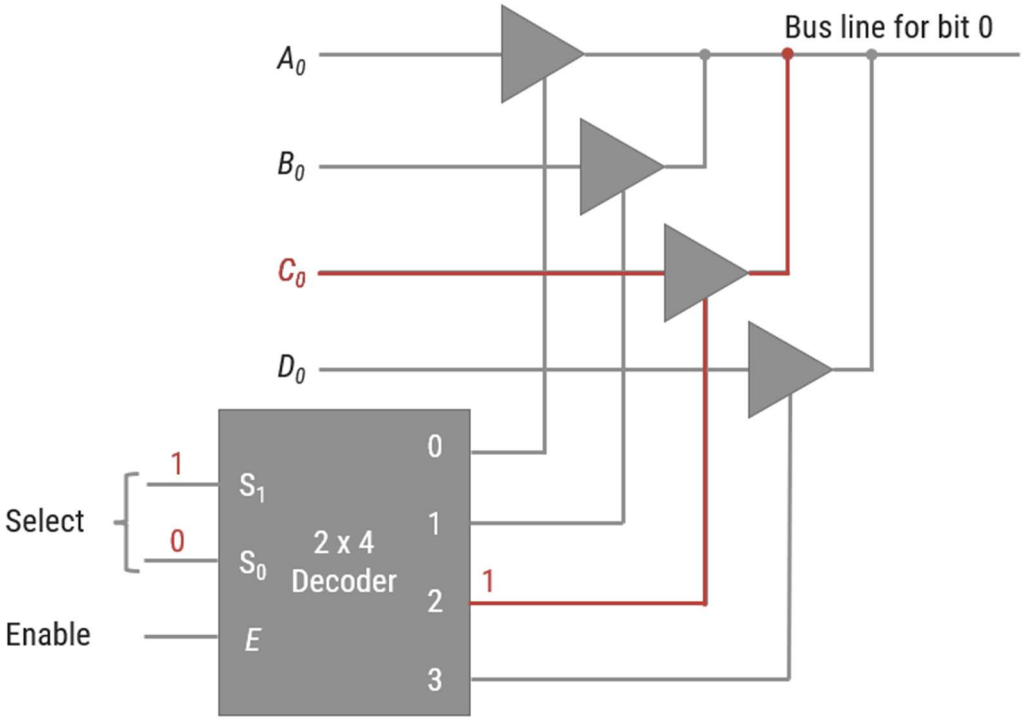
Unit-1

1 What do you mean by register transfer? Explain in detail. Also discuss three-state bus buffer.

- Information transfer from one register to another is designated in symbolic form by means of a replacement operator is known as Register Transfer.
- The term *"register transfer"* implies the availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register.



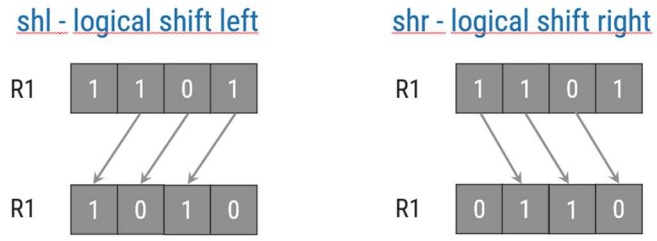
- The outputs of four buffers are connected together to form a single bus line.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high impedance state.
- One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the figure: Bus line with three state-buffers.
- When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.

Name: Sojitra Hemanshu H

2. List and explain types of shift operations on accumulator.

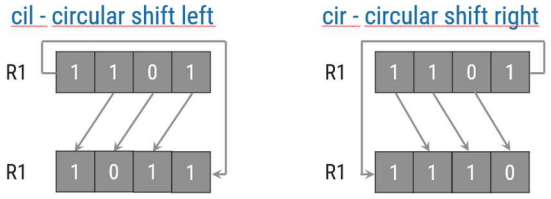
1. Logical Shift

- A *logical shift* is one that transfers 0 through the serial input.



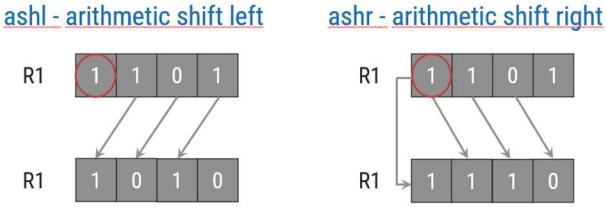
2. Circular Shift

- A *circular shift* (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input.



3. Arithmetic Shift

- 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.



Name: Sojitra Hemanshu H

3. Define RTL. Explain how register transfer takes place in basic computer system

- The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.
- A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.

How it works:

1. **Register Selection:** The instruction specifies the source and destination registers involved in the transfer.
2. **Data Transfer:** The data from the source register is placed on a common bus or internal data path.
3. **Storage:** The data is then stored in the destination register.

4. What is multiplexing? Explain the multiplexing of control signals in ALU.

- Multiplexing is a technique used to transmit multiple signals over a single channel. It allows for efficient utilization of resources by combining multiple data streams into a single stream.

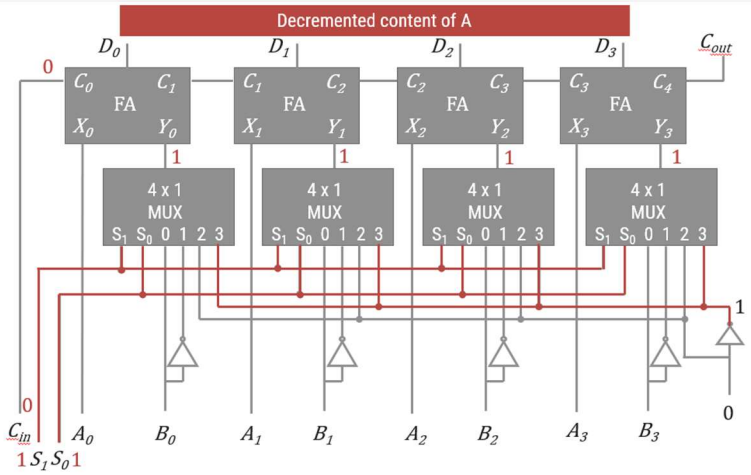
Multiplexing of Control Signals in ALU

In an ALU (Arithmetic Logic Unit), multiplexing is used to select the appropriate control signals based on the operation being performed. This allows the ALU to perform a variety of arithmetic and logical operations using a common set of hardware components.

Here's a simplified explanation of how multiplexing works in an ALU:

1. **Control Signal Selection:** The control unit determines the operation to be performed based on the opcode of the instruction.
2. **Control Signal Generation:** The control unit generates a set of control signals that specify the operation to be performed by the ALU.
3. **Multiplexer:** A multiplexer is used to select the appropriate control signals based on the operation code. The multiplexer has multiple inputs (one for each possible operation) and a single output that connects to the ALU.
4. **ALU Execution:** The selected control signals are sent to the ALU, which performs the specified operation on the input operands.

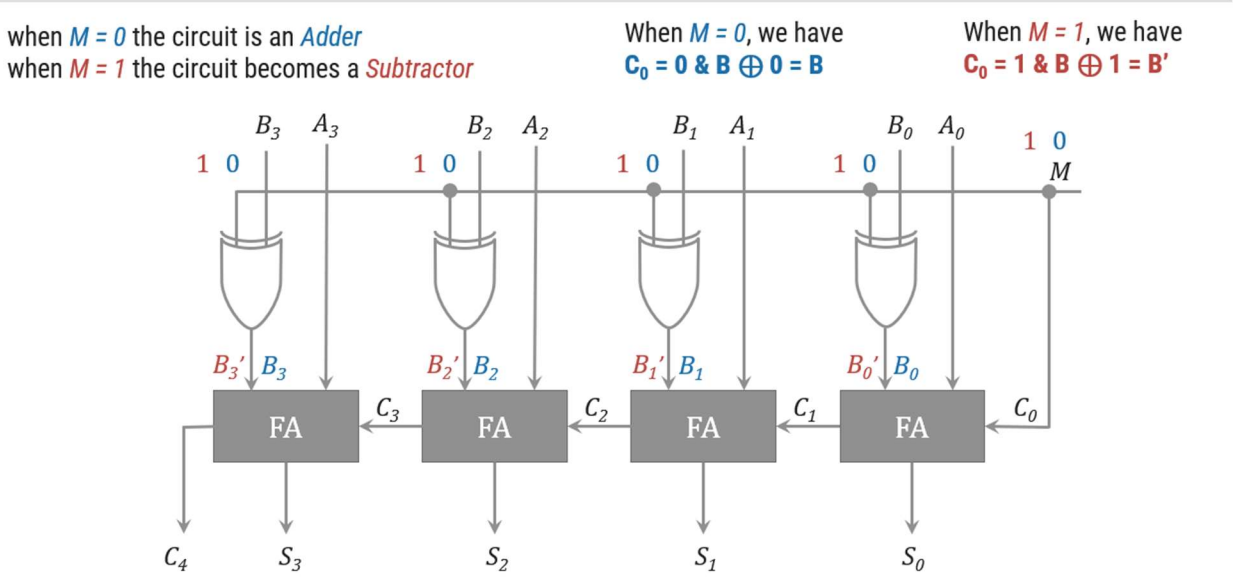
6. Draw the block diagram of 4-bit arithmetic circuit and explain it in detail.



- 4 full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations.
- There are two 4-bit inputs A and B.
- The four inputs from A go directly to the X inputs of the binary adder. Each of the four inputs from B is connected to the data inputs of the multiplexers. The multiplexer's data inputs also receive the complement of B.
- The other two data inputs are connected to logic-0 and logic-1.
    - Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits)
    - Logic-1 signal can be generated through an inverter whose input is 0.
- The four multiplexers are controlled by two selection inputs, S_1 and S_0.
- The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
- 4-bit output $D_0 \ldots D_3$

Name: Sojitra Hemanshu H

9. Explain 4-bit adder-subtractor with diagram.



A 4-bit binary adder-subtractor is a digital circuit capable of performing both addition and subtraction operations on 4-bit binary numbers. It combines the functionality of a 4-bit adder and a 4-bit subtractor.

Basic Structure:

The adder-subtractor typically consists of:

- **Full adders:** Four full adders are connected in series to perform the addition or subtraction.
- **Control signal:** A control signal (often denoted as M) determines whether the circuit performs addition or subtraction.

Operation:

1. **Addition:**
    - When M is 0, the circuit acts as a 4-bit adder.
    - The carry-in input to the first full adder is set to 0.
    - The sum and carry outputs of each full adder are used to generate the final result.
2. **Subtraction:**
    - When M is 1, the circuit acts as a 4-bit subtractor.
    - The carry-in input to the first full adder is set to 1.
    - The sum and carry outputs of each full adder are used to generate the difference and borrow.

Name: Sojitra Hemanshu H

14. Explain selective set, selective complement and selective clear.(if anyone is 1 then 1)

1. Selective Set Operation

- The *selective-set* operation sets to 1 the bits in register A where there are corresponding 1's in register B.
- It does not affect bit positions that have 0's in B.
- **The OR microoperation can be used to selectively set bits of a register.**

1 0 1 0	*A* before
1 1 0 0	*B* (logic operand)
1 1 1 0	*A* after

2. Selective Complement Operation(no similar 1)

- The *selective-complement* operation complements bits in register A where there are corresponding 1's in register B.
- It does not affect bit positions that have 0's in B.
- The exclusive - OR microoperation can be used to selectively set bits of a register.

1 0 1 0	*A* before
1 1 0 0	*B* (logic operand)
0 1 1 0	*A* after

3. Selective Clear Operation(1 only if it is in A if 1 is in B then It makes whole 0)

- The *selective-clear* operation clears to 0 the bits in register A only where there are corresponding 1's in register B.
- It does not affect bit positions that have 0's in B.
- The corresponding logic microoperation is A ← A ∧ B'.

1 0 1 0	*A* before
1 1 0 0	*B* (logic operand)
0 0 1 0	*A* after

Name: Sojitra Hemanshu H

16. Explain Micro operation.

- Logic micro operations specify binary operations for strings of bits stored in registers.
- These operations consider each bit of the register separately and treat them as binary variables.
- It specifies a logic microoperation to be executed on the individual bits of the registers provided that the control variable P = 1
- The logic microoperations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.
- Special symbols will be adopted for the logic microoperations OR, AND, and complement, to distinguish them from the corresponding symbols used to express Boolean functions.
- The symbol V will be used to denote an OR microoperation and the symbol ^ to denote an AND microoperation. The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.
- By using different symbols, it will be possible to differentiate between a logic microoperation and a control (or Boolean) function.
- The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
- Although there are 16 logic microoperations, most computers use only four-AND, OR, XOR (exclusive-OR), and complement from which all others can be derived.

19. What is a Bus?

A bus in computer architecture is a communication system that allows different components of a computer to exchange data. It's like a highway that connects various parts of the computer, enabling them to communicate and share information.

There are several types of buses in a computer system:

1. **Data bus:** This bus is used to transfer data between the processor, memory, and other components. It's bidirectional, meaning data can flow in both directions.
2. **Address bus:** This bus is used to specify the memory address that the processor wants to access. It's unidirectional, as the address flows from the processor to the memory.
3. **Control bus:** This bus is used to carry control signals between the processor and other components. These signals coordinate the activities of the various components.

The width of a bus (measured in bits) determines the amount of data that can be transferred at once. A wider bus allows for faster data transfer rates.

20. What is an ALU

ALU (Arithmetic Logic Unit) is a crucial component of a computer's Central Processing Unit (CPU). It's responsible for performing arithmetic and logical operations on data.

Key Functions:

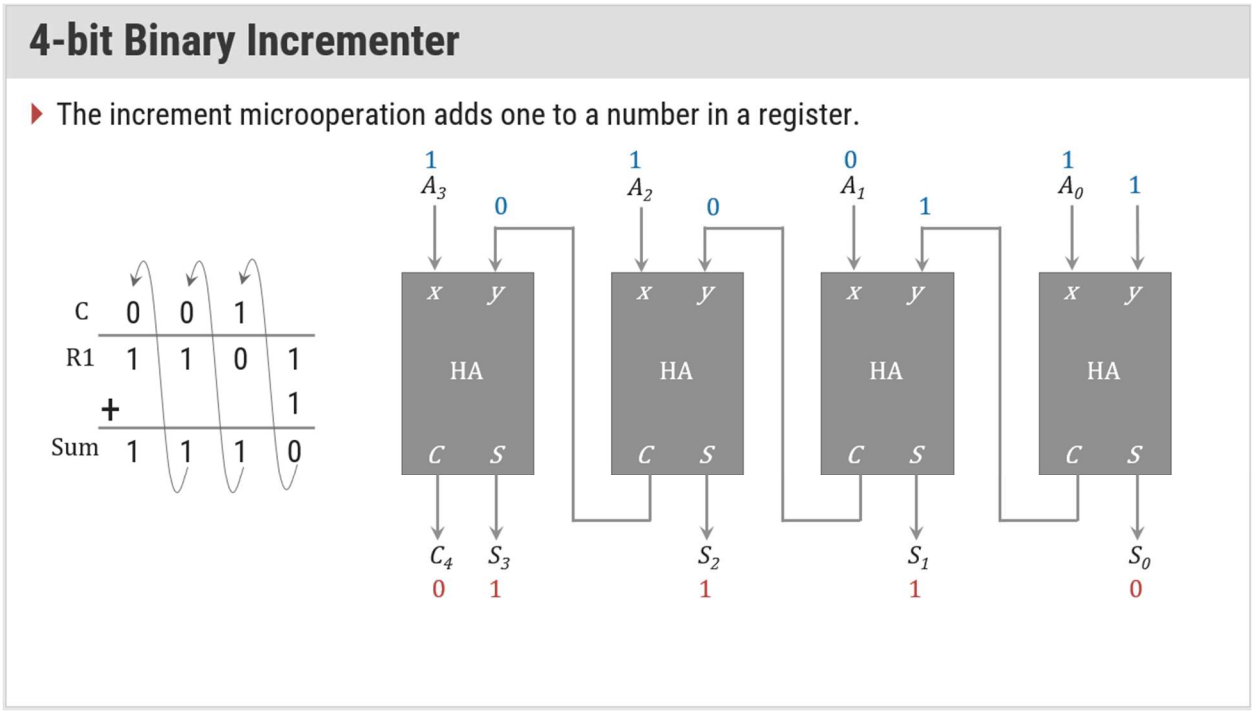
- **Arithmetic Operations:**
    - Addition
    - Subtraction
    - Multiplication
    - Division
    - Increment/Decrement
- **Logical Operations:**
    - AND
    - OR
    - NOT
    - XOR
    - Shift operations (left/right shift)

How it works:

1. **Input:** The ALU receives operands (data to be operated on) from registers or memory.
2. **Operation:** The control unit signals the ALU to perform a specific operation based on the instruction being executed.
3. **Calculation:** The ALU performs the calculation using its internal circuitry.
4. **Output:** The result of the calculation is stored back in a register or memory location.

In essence, the ALU is the brain of the CPU, carrying out the core computations that make a computer function. The efficiency and speed of an ALU significantly impact the overall performance of a processor.

23. Draw a diagram of 4-bit binary incrementer and explain it briefly



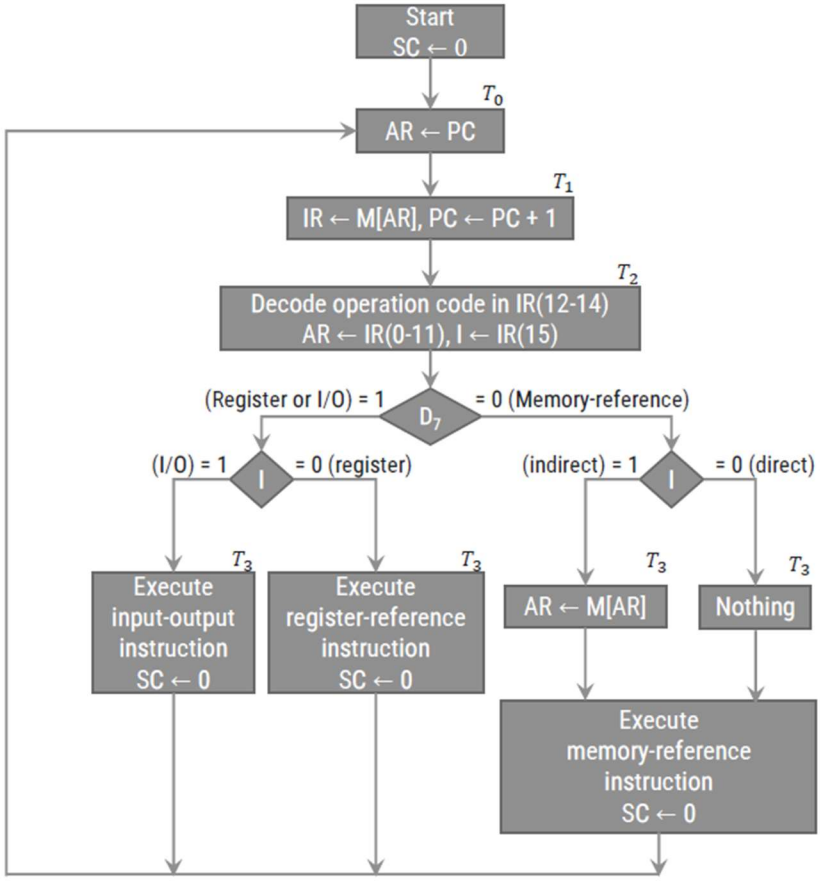
Here's how it works:

1. **Input:** The 4-bit binary number to be incremented is provided as input to the first full adder.
2. **Full Adder:** The full adder takes three inputs: the current bit from the input number, the carry-in from the previous stage (which is 0 for the first stage), and the carry-out from the previous stage (which is initially 0). It produces two outputs: the sum bit and the carry-out bit.
3. **Carry Propagation:** The carry-out bit from the first full adder is connected to the carry-in input of the second full adder, and so on for the remaining stages.
4. **Output:** The sum bits from all the full adders together form the incremented 4-bit binary number.

Name: Sojitra Hemanshu H

Unit -2

1. Write a detailed note on instruction cycle with neat diagrams.



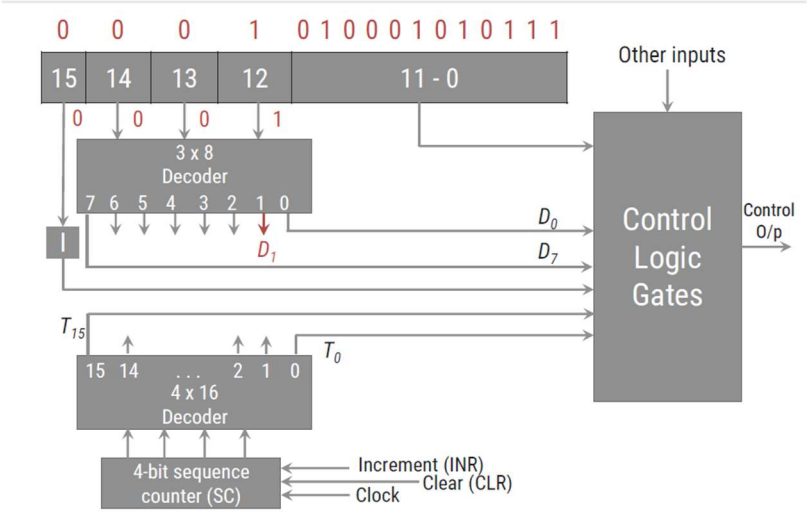
Name: Sojitra Hemanshu H

Instruction Cycle

- A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:
  1. Fetch an instruction from memory.
  2. Decode the instruction.
  3. Read the effective address from memory if the instruction has an indirect address.
  4. Execute the instruction.
- After step 4, the control goes back to step 1 to fetch, decode and execute the next instruction.
- This process continues unless a HALT instruction is encountered.

- Fetch & Decode
  - PC is loaded with the address of the first instruction in the program.
  - The micro-operations for fetch and decode phases are as follows:
- $T_0 : AR \leftarrow PC$
- $T_1 : IR \leftarrow M[AR], PC \leftarrow PC + 1$
- $T_2 : D_0, \ldots, D_7 \leftarrow Decode\ IR(12-14),\ AR \leftarrow IR(0-11),\ I \leftarrow IR(15)$
- Determine the type of instruction
  - During time T_3, the control unit determines the type of instruction i.e. Memory reference, Register reference or Input-Output instruction.
  - If $D_7 = 1$ then instruction must be register reference or input-output else memory reference instruction.

2. Explain control unit of basic computer and its working with diagram.



- Components of Control unit are

1. Two decoders

2. A sequence counter

3. Control logic gates

- An instruction read from memory is placed in the instruction register (IR).
- In control unit the IR is divided into three parts: I bit, the operation code (12-14)bit, and bits 0

Name: Sojitra Hemanshu H

through 11.

- The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder.
- Bit-15 of the instruction is transferred to a flip-flop designated by the symbol I.
- The eight outputs of the decoder are designated by the symbols D0 through D7.
- Bits 0 through 11 are applied to the control logic gates.
- The 4‐bit sequence counter can count in binary from 0 through 15. The outputs of counter are

decoded into 16 timing signals T0 through T15.

- The sequence counter SC can be incremented or cleared synchronously.

3. For the basic computer explain following instructions

1. LDA

1. LDA (Load Accumulator):

- **Purpose:** Loads a value from memory into the accumulator register.
- **Format:** LDA address
- **Example:** LDA 100H (loads the value at memory location 100H into the accumulator)

2. ADD

2. ADD:

- **Purpose:** Adds the contents of the accumulator to a specified register or memory location.
- **Format:** ADD register/address
- **Example:** ADD R1 (adds the contents of R1 to the accumulator) or ADD 200H (adds the value at memory location 200H to the accumulator)

3. AND

3. AND:

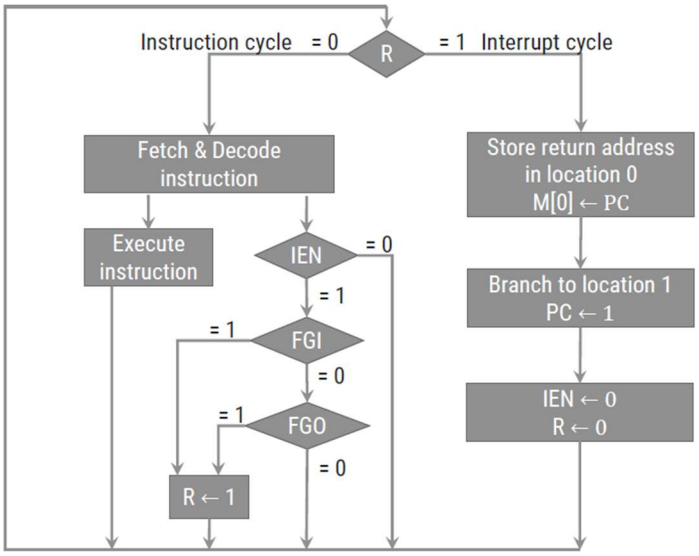
- **Purpose:** Performs a bitwise AND operation on the accumulator and a specified register or memory location.
- **Format:** AND register/address
- **Example:** AND R2 (performs a bitwise AND operation between the accumulator and R2)

4. CLA

4. CLA (Clear Accumulator):

- **Purpose:** Sets the accumulator to zero.
- **Format:** CLA

4. Draw and explain flowchart for interrupt cycle.



The interrupt cycle is a hardware implementation of a branch and save return address operation.

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

Name: Sojitra Hemanshu H

5. For the basic computer explain following instructions

1. BUN

1. BUN (Branch Unconditionally):

- **Purpose:** Transfers control to a specified address unconditionally.
- **Explanation:** This instruction is used to jump to a specific location in the program, regardless of any conditions. It's often used for implementing loops, subroutines, or unconditional jumps.

2. BSA

2. BSA (Branch and Save Return Address):

- **Purpose:** Transfers control to a specified address and saves the current program counter (PC) in a designated register or memory location.
- **Explanation:** This instruction is used for subroutine calls. It saves the return address so that the program can return to the correct location after the subroutine finishes executing.

3. CIL

3. CIL (Compare Immediate with Link):

- **Purpose:** Compares an immediate value with the contents of a register and sets a condition code based on the result. If the condition is met, a branch to a specified address occurs.
- **Explanation:** This instruction is used for conditional branching. It allows the program to make decisions based on the comparison result. For example, it can be used to implement loops or conditional statements.

4. SZE

4. SZE (Skip if Zero):

- **Purpose:** Skips the next instruction if the zero flag (Z) is set.
- **Explanation:** The zero flag is set if the result of the previous arithmetic or logical operation was zero. This instruction can be used to implement conditional branching based on the result of a comparison or calculation.

7. State the differences between hardwired control and microprogrammed control.

Hardwired Control

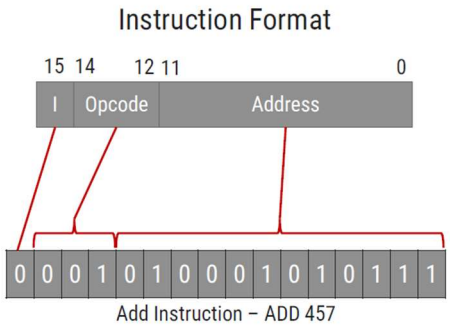
- **Implementation:** Uses a combination of logic gates and flip-flops to create a fixed sequence of control signals.
- **Flexibility:** Less flexible, as changing the control logic requires modifying the hardware.
- **Performance:** Generally faster due to its direct implementation.
- **Design:** More complex to design and implement due to the intricate interconnection of logic gates.

Name: Sojitra Hemanshu H

Microprogrammed Control

- **Implementation:** Uses a read-only memory (ROM) to store a sequence of microinstructions. These microinstructions define the control signals needed for each instruction.
- **Flexibility:** More flexible, as changes to the control logic can be made by modifying the microprogram.
- **Performance:** Slightly slower due to the additional memory access required to fetch microinstructions.
- **Design:** Easier to design and modify compared to hardwired control.

9. Draw and explain basic computer instruction formats.



The image you provided illustrates a **fixed-length instruction format** with a **16-bit** word size. Here's a breakdown of the different fields:

1. I (Instruction Type):

- This bit indicates whether the instruction is an immediate addressing mode instruction (I=1) or a direct addressing mode instruction (I=0).

2. Opcode:

- This field contains the operation code that specifies the type of operation to be performed (e.g., ADD, SUB, LOAD, STORE).
- In the example, the opcode is 0010, which might represent the "ADD" instruction.

3. Address:

- This field contains the memory address or register operand for the instruction.
- In the example, the address is 01010111 (decimal 457).

10. Differentiate MRI and non-MRI.

Memory-Reference Instructions

- **Access memory:** These instructions directly access memory to read or write data.
- **Addressing modes:** They use various addressing modes (e.g., direct, indirect, indexed) to specify the memory location.
- **Memory-Reference:** Often have a larger opcode field to accommodate the addressing mode information.
- **Memory-Reference:** Can be slower due to the additional memory access time.
- **Examples:**
    - **LOAD:** Loads data from memory into a register.
    - **STORE:** Stores data from a register into memory.
    - **ADD:** Adds a value from memory to a register.

Non-Memory-Reference Instructions

- **Operate on registers:** These instructions primarily operate on data stored in registers.
- **No memory access:** They do not directly access memory.
- **Non-Memory-Reference:** May have a smaller opcode field due to the reduced complexity of addressing.
- **Non-Memory-Reference:** Generally faster, as they avoid memory accesses.
- **Examples:**
    - **MOV:** Moves data between registers.
    - **ADD:** Adds two register values.
    - **AND:** Performs a bitwise AND operation on two registers.

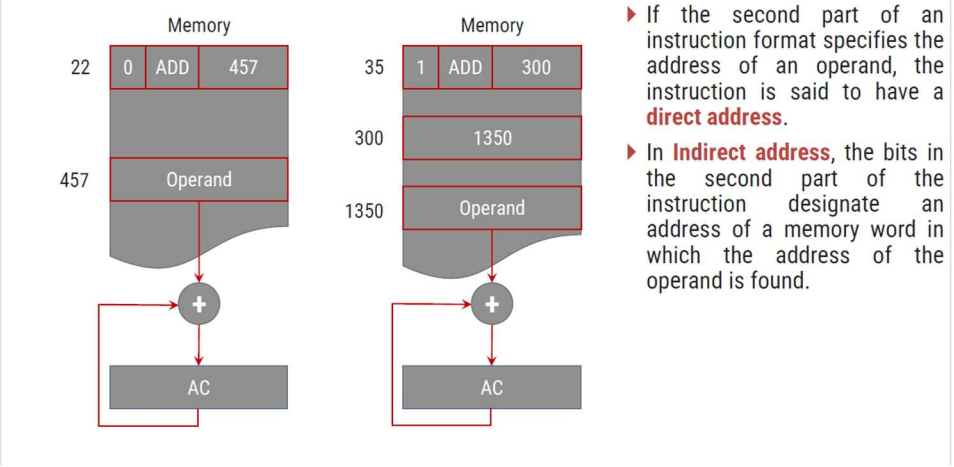
11. Explain Direct and Indirect Addressing.

・ it is sometimes convenient to use the address bits of an instruction code not as an address but as the actual operand. When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand.

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

・ This is in contrast to a third possibility called indirect address, where the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.

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



- If the second part of an instruction format specifies the address of an operand, the instruction is said to have a **direct address**.
- In **Indirect address**, the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.

13. What is Interrupt? How it is useful for a system?

An interrupt is a signal that causes a computer to temporarily stop its current task and switch to a predefined routine to handle a specific event or condition. Interrupts can be generated by hardware devices (like a keyboard or mouse), software events (like a division by zero), or external signals.

How Interrupts Are Useful:

1. **Efficient Handling of Events:** Interrupts allow the system to respond quickly to external events without having to constantly poll for changes. This improves system responsiveness and efficiency.
2. **Time-Critical Tasks:** Interrupts can be used to handle time-critical tasks, such as real-time input/output operations or communication with external devices.
3. **Asynchronous Events:** Interrupts can be used to handle asynchronous events, which occur at unpredictable times.
4. **Modular Design:** Interrupts can help create modular and flexible system designs by separating the main program from interrupt handlers.
5. **Error Handling:** Interrupts can be used to handle errors and exceptions that occur during program execution.

14. Explain CLA, ISZ, INP instruction.

CLA (Clear Accumulator)

- **Purpose:** Sets the accumulator register to zero.
- **Syntax:** CLA
- **Functionality:** Resets the accumulator to its initial state, clearing any previous value.

ISZ (Increment and Skip if Zero)

ISZ: Increment and Skip if Zero

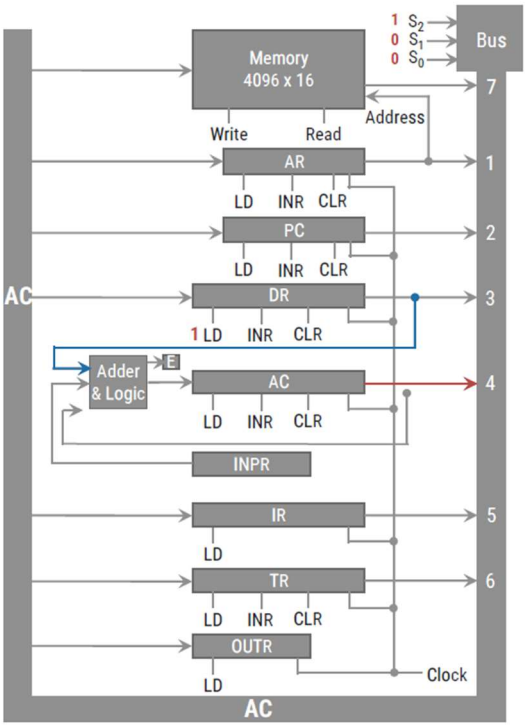
These instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1. Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.

D_6T_4: DR ← M[AR]
D_6T_5: DR ← DR + 1
D_6T_6: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

INP (Input)

- **Purpose:** Reads a value from an input device and stores it in the accumulator.
- **Syntax:** INP port
- **Functionality:** The port parameter specifies the input port number. The instruction reads a value from the specified port and stores it in the accumulator.

15. Explain seven register common bus system.



- **Registers:** The diagram shows seven registers: AC (Accumulator), AR (Address Register), PC (Program Counter), DR (Data Register), IR (Instruction Register), TR (Temporary Register), and OUTR.
- **Common Bus:** The vertical line labeled "Bus" represents the common bus that connects all the registers. This allows data to be transferred between registers efficiently.
- **Memory:** The memory unit is shown with a capacity of 4096 x 16 bits. This means it can store 4096 words of 16 bits each.
- **Control Signals:** The lines labeled "Write," "Read," "LD" (Load), "INR" (Increment), and "CLR" (Clear) represent control signals that govern the operations of the system.

How it Works:

1. **Fetch:** The PC holds the address of the next instruction to be executed. This address is placed on the address bus, and the instruction is fetched from memory and stored in the IR.
2. **Decode:** The opcode of the instruction is decoded to determine the operation to be performed and the operands involved.
3. **Execute:** The specified operation is executed using the registers and the common bus. For example, to add two numbers, the operands would be transferred to the adder and logic unit, and the result would be stored in a register.
4. **Write Back:** If necessary, the result of the operation may be written back to memory or another register.

17. What is a Program Counter?

In computer architecture, the Program Counter (PC) is a special-purpose register that holds the memory address of the next instruction to be executed. It acts as a pointer, guiding the processor through the sequence of instructions in a program.

Key Functions:

- **Instruction Fetch:** The PC provides the address to the memory control unit, which fetches the instruction stored at that location.
- **Sequencing:** After an instruction is executed, the PC is updated to point to the next instruction in the program.
- **Branching:** When a branch instruction is encountered, the PC is modified to jump to a different location in the program.
- **Interrupt Handling:** During an interrupt, the PC is saved to remember the current execution point, allowing the program to resume execution after the interrupt is handled.

18. What is an Accumulator?

In computer architecture, an accumulator is a special-purpose register that is often used for arithmetic operations. It holds one of the operands for an operation and also stores the result of the operation.

Key Characteristics:

- **Central Role:** The accumulator plays a central role in many arithmetic and logical operations.
- **Implicit Operand:** In some architectures, the accumulator is an implicit operand for certain instructions, meaning it is not explicitly specified in the instruction.
- **Efficiency:** Using the accumulator can sometimes improve the efficiency of instruction execution, as it eliminates the need to specify a register for one of the operands.

19. What is an Instruction Register?

In computer architecture, the Instruction Register (IR) is a special-purpose register that holds the current instruction being executed by the processor. It acts as a buffer between the instruction fetch stage and the decode and execute stages of the instruction cycle.

Key Functions:

- **Instruction Storage:** The IR stores the fetched instruction from memory.
- **Decoding:** The control unit decodes the opcode of the instruction stored in the IR to determine the operation to be performed and the operands involved.
- **Execution:** The decoded instruction is executed based on the contents of the IR.

20. What do you understand by Memory Address?

In computer architecture, a memory address is a unique numerical identifier that specifies the location of a specific byte or word of data within a computer's memory. It's like a postal address that tells the computer where to find the data it needs.

Key Points:

- **Unique Identifier:** Each memory location has a unique address.
- **Byte or Word:** Addresses typically refer to the location of a byte or word of data.
- **Sequential:** Memory addresses are typically sequential, meaning they increase in numerical order.
- **Used by Instructions:** Instructions often use memory addresses to specify the location of data to be accessed or modified.

21. What is a Carry Flag?

In computer architecture, the Carry Flag (CF) is a special-purpose bit in the processor's status register that indicates whether an arithmetic operation has resulted in a carry or borrow.

Purpose:

- **Overflow Detection:** The CF is used to detect overflow conditions in arithmetic operations, such as addition and subtraction. When the result of an operation exceeds the maximum representable value for the data type, the CF is set.
- **Shift Operations:** The CF can be used to store the bit that is shifted out during shift operations.
- **Conditional Branches:** The CF can be used in conditional branching instructions to determine whether a comparison or arithmetic operation resulted in a specific condition (e.g., greater than, less than, equal to).

22. Explain Instruction Fetch.

Instruction Fetch is the first stage of the instruction cycle in a computer's central processing unit (CPU). It involves retrieving the next instruction to be executed from memory.

Here's how it works:

1. **Program Counter (PC):** The PC holds the memory address of the next instruction to be fetched.
2. **Memory Access:** The PC is sent to the memory unit, which retrieves the instruction stored at that address.
3. **Instruction Register (IR):** The fetched instruction is loaded into the IR.

24. Enlist major components of CPU.

 Control Unit (CU):

- Responsible for coordinating the activities of the CPU's other components.
- Fetches instructions from memory, decodes them, and sends control signals to execute them.

 Arithmetic Logic Unit (ALU):

- Performs arithmetic and logical operations on data.
- Handles calculations, comparisons, and bitwise operations.

 Registers:

- Small, high-speed storage units used to hold data temporarily during processing.
- Common registers include:
    - **General-purpose registers:** Used for various purposes.
    - **Special-purpose registers:** Used for specific tasks (e.g., program counter, instruction register, status register).

 Clock:

- Generates a regular sequence of pulses that synchronize the operation of the CPU's components.

 Interconnect:

- A network of wires or buses that connects the various components of the CPU and allows them to communicate.

25. Effective address.

In computer architecture, an effective address is the actual memory address that is used to access data during an instruction execution. It is often calculated based on the base address specified in the instruction and any additional addressing modes that may be used.

Addressing Modes:

- **Direct Addressing:** The effective address is equal to the address specified in the instruction.
- **Indirect Addressing:** The effective address is the value stored at the memory location specified in the instruction.
- **Indexed Addressing:** The effective address is calculated by adding a base address and an index register.
- **Relative Addressing:** The effective address is calculated by adding a displacement value to the program counter (PC).
- **Based Addressing:** The effective address is calculated by adding a base register and an offset.