Computer Organization (BCA III Semester)

Unit – I

A. 1. What is Data representation explain in details?

Binary System:
In the binary system, each digit (or bit) can only be 0 or 1.
A group of eight bits is called a byte, and it is a fundamental unit of storage.

Hexadecimal Notation:
Binary representation can be cumbersome for humans, so hexadecimal (base-16) notation is often used to represent binary patterns more compactly.
Each hexadecimal digit represents four bits (half a byte).

Data Types:
Different types of data (e.g., integers, floating-point numbers, characters) are represented using specific data types.
Integer values are usually represented using binary or two's complement notation.
Floating-point numbers are represented using a sign bit, exponent, and fraction (mantissa) in a specific format like IEEE 754.

Character Encoding:
Characters are represented using character encodings, such as ASCII (American Standard Code for Information Interchange) or Unicode.
ASCII uses 7 or 8 bits to represent characters, while Unicode typically uses 16 or 32 bits.

Memory Organization:
Memory in a computer is organized into addressable units, usually bytes.
Each byte in memory has a unique address, allowing the computer to retrieve and store data.

Endianness:
Endianness refers to the byte order in multi-byte data types.
In a big-endian system, the most significant byte is stored at the lowest memory address.
In a little-endian system, the least significant byte is stored at the lowest memory address.
Representation of Images and Sound:
Images and sound are represented using different formats, such as JPEG for images and MP3 for audio.
These formats use compression techniques to reduce the amount of data needed for storage.

Q.2.what is data type explain in details?

Ans:- Data types in computer programming define the kind of values a variable can hold and the operations that can be performed on those values.

Data types are divided into two groups:

. Primitive data types - includes byte, short, int, long, float, double, boolean and char

. Non-primitive data types - such as String, Arrays and Classes (you will learn more about these in a later chapter)

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types :

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits

double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Q. 3. What is Number System and its Types?

Ans:- The number system is simply a system to represent or express numbers. There are various types of number systems and the most commonly used ones are decimal number system, binary number system, octal number system, and hexadecimal number system.

Number System Conversion Table

Binary Numbers	Octal Numbers	Decimal Numbers	Hexadecimal Numbers
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8

1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

Decimal to Other Bases

Converting a decimal number to other base numbers is easy. We have to divide the decimal number by the converted value of the new base.

Decimal to Binary Number:

Suppose if we have to convert decimal to binary, then divide the decimal number by 2.

Example 1. Convert (25)10 to binary number.

Solution: Let us create a table based on this question.

Operation	Output	Remainder
25 ÷ 2	12	1(MSB)
12 ÷ 2`	6	0
6 ÷ 2	3	0
3 ÷ 2	1	1

1 ÷ 2	0	1(LSB)

Therefore, from the above table, we can write,

$(25)_{10} = (11001)_2$

Decimal to Octal Number:

To convert decimal to octal number we have to divide the given original number by 8 such that base 10 changes to base 8. Let us understand with the help of an example.

Example 2: Convert 128_{10} to octal number.

Solution: Let us represent the conversion in tabular form.

Operation	**Output**	**Remainder**
128÷8	16	0(MSB)
16÷8	2	0
2÷8	0	2(LSB)

Therefore, the equivalent octal number = 200_8

Decimal to Hexadecimal:

Again in decimal to hex conversion, we have to divide the given decimal number by 16.

Example 3: Convert 128_{10} to hex.

Solution: As per the method, we can create a table;

Operation	Output	Remainder
128÷16	8	0(MSB)
8÷16	0	8(LSB)

Therefore, the equivalent hexadecimal number is 80_{16}

Here MSB stands for a Most significant bit and LSB stands for a least significant bit.

Other Base System to Decimal Conversion

Binary to Decimal:

In this conversion, binary number to a decimal number, we use multiplication method, in such a way that, if a number with base n has to be converted into a number with base 10, then each digit of the given number is multiplied from MSB to LSB with reducing the power of the base. Let us understand this conversion with the help of an example.

Example 1. Convert $(1101)_2$ into a decimal number.

Solution: Given a binary number $(1101)_2$.

Now, multiplying each digit from MSB to LSB with reducing the power of the base number 2.

$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 8 + 4 + 0 + 1$

$= 13$

Therefore, $(1101)_2 = (13)_{10}$

Octal to Decimal:

To convert octal to decimal, we multiply the digits of octal number with decreasing power of the base number 8, starting from MSB to LSB and then add them all together.

Example 2: Convert 228 to decimal number.

Solution: Given, 22_8

$2 \times 8^1 + 2 \times 8^0$

$= 16 + 2$

$= 18$

Therefore, $22_8 = 18_{10}$

Hexadecimal to Decimal:

Example 3: Convert 121_{16} to decimal number.

Solution: $1 \times 16^2 + 2 \times 16^1 + 1 \times 16^0$

$= 16 \times 16 + 2 \times 16 + 1 \times 1$

$= 289$

Therefore, $121_{16} = 289_{10}$

Hexadecimal to Binary Shortcut Method

To convert hexadecimal numbers to binary and vice versa is easy, you just have to memorize the table given below.

Hexadecimal Number	Binary
0	0000
1	0001

2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

You can easily solve the problems based on hexadecimal and binary conversions with the help of this table. Let us take an example.

Example: Convert $(89)_{16}$ into a binary number.

Solution: From the table, we can get the binary value of 8 and 9, hexadecimal base numbers.

8 = 1000 and 9 = 1001

Therefore, $(89)_{16} = (10001001)_2$

Octal to Binary Shortcut Method

To convert octal to binary number, we can simply use the table. Just like having a table for hexadecimal and its equivalent binary, in the same way, we have a table for octal and its equivalent binary number.

Octal Number	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Example: Convert $(214)_8$ into a binary number.

Solution: From the table, we know,

2 → 010

1 → 001

4 → 100

Therefore,(214)8 = (010001100)2

Q.4.Write short notes on following topics:-

A. Binary Addition & Subtraction B. Binary Multiplication & division

Ans:- ## Rules of Binary Addition

Binary addition is much easier than the decimal addition when you remember the following tricks or rules. Using these rules, any binary number can be easily added. The four rules of binary addition are:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 =10

How To Do Binary Addition?

Now, look at the example of the binary addition:101 + 101

Procedure for Binary Addition of Numbers:

101

(+) 101

- **Step 1:** First consider the 1's column, and add the one's column,(1+1) and it gives the result 10 as per the condition of binary addition.
- **Step 2:** Now, leave the 0 in the one's column and carry the value 1 to the 10's column.

1

101

(+) 101

————————

0

- **Step 3: Now add 10’s place, 1+(0 + 0) = 1.** So, nothing carries to the 100’s place and leave the value 1 in the 10’s place

1

101

(+) 101

————————-

10

- **Step 4:** Now add the 100’s place (1 + 1) = 10. Leave the value 0 in the 100’s place and carries 1 to the 1000’s place.

1

101

(+) 101

————————-

1010

So, the resultant of the addition operation is 1010.

When you cross-check the binary value with the decimal value, the resultant value should be the same.

The binary value 101 is equal to the decimal value 5

So, 5 + 5 = 10

The decimal number 10 is equal to the binary number 1010.

Binary Addition Table

The table of adding two binary numbers 0 and 1 is given below:

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	0 (where 1 is carried over)

You can see from the above table, x and y are the two binary numbers. So when we give the input for x = 0 and y = 0, then the output is equal to 0. When x = 0 or 1 and y = 1 or 0, then x+y = 1. But when both x and y are equal to 1, then their addition equals to 0, but the carryover number will equal to 1, which means basically 1 + 1 = 10 in binary addition, where 1 is carry forwarded to the next digit.

Examples of Binary Addition

A few examples of binary additions are as follows:

Example 1: 10001 + 11101

Solution:

1

1 0 0 0 1

(+) 1 1 1 0 1

_____

1 0 1 1 1 0

Example 2: 10111 + 110001

Solution:

1 1 1

1 0 1 1 1

(+) 1 1 0 0 0 1

———————

1 0 0 1 0 0 0

Binary Subtraction Rules

Rules and tricks: Binary subtraction is much easier than the decimal subtraction when you remember the following rules:

- 0 – 0 = 0
- 0 – 1 = 1 (with a borrow of 1)
- 1 – 0 = 1
- 1 – 1 = 0

Now, look at the example of the binary subtraction: 101 from 1010

How to Subtract Binary Numbers?

Learn how to do binary subtraction using the example: 1010 – 101

Procedure to do Binary Subtraction:

1010

(-) 101

Step 1: First consider the 1's column, and subtract the one's column,(0 – 1) and it gives the result 1 as per the condition of binary subtraction with a borrow of 1 from the 10's place.

Step 2: After borrowing 1 from the 10’s column, the value 1 in the 10’s column is changed into the value 0

1 Borrow

1 0 1 0

(-) 1 0 1

_____

1

Step 3: So, subtract the value in the 10’s place, (0 – 0) = 0.

1 Borrow

1 0 1 0

(-) 1 0 1

_____

0 1

Step 4: Now subtract the values in 100’s place. Borrow 1 from the 1000’s place (0 – 1) = 1.

1 1 Borrow

1 0 1 0

(-) 1 0 1

_____

0 1 0 1

So, the resultant of the subtraction operation is 0101.

When you cross-check the binary subtraction resultant value with the decimal value, the resultant value should be the same.

The binary value 1010 is equal to the decimal value 10, and 101 is equivalent to 5

So, 10 – 5 = 5

Therefore, the decimal number 5 is equal to the binary number 0101.

Example 1: 0011010 – 001100

Solution:

1 1 Borrow

0 0 1 1 0 1 0

(-) 0 0 1 1 0 0

_____

0 0 0 1 1 1 0

Decimal Equivalent :

0 0 1 1 0 1 0 = 26

0 0 1 1 0 0 = 12

Therefore, 26 – 12 = 14

The binary resultant 0 0 0 1 1 1 0 is equivalent to 14.

Unit – II

Q.1. What is Register transfer language :-

Ans:- Register transfer language is a symbolic notation for describing micro-operation transfers between registers.

The availability of hardware logic circuits that can perform a specified micro-operation and transfer the outcome of the operation to the same or another register is referred to as register transfer. The term "language" was coined by programmers to describe programming languages. This programming language is a method of expressing a computer process through symbols.

Following are some commonly used register transfer example with an example:

1. Accumulator: This is the most commonly used register for storing data read from memory.

2. General-Purpose Registers: These are used to store data on intermediate outcomes during the execution of a programme. Assembly programming is required to access it.

3. Special Purpose Registers: Users do not have access to the Special Purpose Registers. These are computer system registers.

MAR: Memory Address Registers are the registers that store the memory unit's address

MBR: This register stores instructions and data received from and sent from the memory

PC: Program Counter indicates the next command to be executed

IR: Instruction Register stores the to-be-executed instruction

Here are a few key points related to RTL:

Registers:

RTL focuses on the flow of data between registers, which are small, fast storage locations in a CPU or digital circuit.

Operations:

Describes the operations and transformations that occur between registers. These operations typically include arithmetic, logic, and data movement operations.

Abstraction Level:

RTL is at a lower level of abstraction compared to high-level programming languages. It is closer to the hardware and provides a detailed representation of the data flow.

Hardware Description Languages (HDLs):

RTL is often expressed using hardware description languages like Verilog or VHDL. These languages allow designers to describe the behavior of digital circuits and systems.

Synthesis:

RTL descriptions can be synthesized into actual hardware by tools called synthesizers. These tools generate a netlist, which is a description of the circuit in terms of logic gates and flip-flops.

Design Verification:

RTL descriptions are used in simulation and verification processes to ensure that the digital design behaves correctly.

It's important to note that while RTL is crucial in the hardware design process, it is not used for general-purpose programming. High-level programming languages like C, Python, or Java are used for software development, while RTL and HDLs are used for describing the hardware at a lower level.

Q. 2.Explainin in details Register transfer Bus and Memory transfer .

Ans: In computer architecture and digital design, the concepts of register transfer, buses, and memory transfers play crucial roles. Let's briefly explore each of these concepts:

1. **Register Transfer:**
   Definition: Register transfer is a fundamental operation in digital systems where data is transferred between registers.

   Explanation: In a digital system, registers are small, fast storage locations that hold data temporarily. Register transfer operations involve moving data between these registers. The data can be transferred within the CPU or between the CPU and external devices.

2. **Bus:**
   Definition: A bus is a communication system that transfers data between components in a computer or between different devices.

   Explanation: Buses consist of a set of wires that carry information (data, address, or control signals) between various parts of a computer. There are different types of buses, including data buses, address buses, and control buses. The data bus is responsible for transferring actual data, the address bus specifies a memory location or an I/O port, and the control bus carries signals that control the operation of the components.

3. **Memory Transfer:**
   Definition: Memory transfer involves the movement of data between the memory and other parts of the computer system.

   Explanation: In a computer system, the memory stores both data and instructions. Memory transfer operations include reading data from memory into registers or writing data from registers into memory. These operations are crucial for the execution of programs and the storage of data.

   Now, let's tie these concepts together:

- **Register Transfer using a Bus:**
  In a computer system, data is often transferred between registers using a bus. The data bus carries the actual data, the address bus specifies the source and destination registers, and the control bus manages the operation.

- **Memory Transfer using a Bus:**
  When transferring data between memory and the CPU, a similar process occurs. The CPU specifies the memory address (using the address bus) from which it wants to read or to which it wants to write data. The data bus is used to transfer the actual data, and the control bus manages the read or write operation.

  In summary, register transfer, buses, and memory transfers are fundamental concepts in digital systems, enabling the movement of data within a computer and between the CPU and memory. These operations are crucial for the proper functioning of a computer system.

Q.3. What is micro-operation explain in details?

Ans: **Micro-Operations**

Micro-operations are operations performed on data stored in registers. A micro-operation is a simple operation that is carried out on data contained in one or more registers.

Example: Load, Shift, count, and clear.

Types of Micro-Operations

The following are the different types of micro-operations:

1. Micro-operations that move binary data from one register to another are known as register transfers.

2. In registers, arithmetic micro-operations operate on numeric data stored.

3. Bit manipulation operations on non-numeric data are performed by logic micro-operations.

4. Shift micro-operations are data-based shift micro-operations.

1. Arithmetic Micro-Operations

· Add Micro-Operation

The following statement defines it:

R1 + R2 = R3

The foregoing line tells the computer to add the data or contents of register R1 to the data or contents of register R2, then transfer the sum to register R3.

· Subtract Micro-Operation

Consider the following scenario:

R1 + R2′ + 1 R3

Instead of using the minus operator, we use the complement of 1 and add one to the register being subtracted.

· Increment/Decrement Micro-Operation

In general, increment and decrement micro-operations are accomplished by adding and removing 1 from the register.

R1 → R1 + 1

R1 → R1 – 1

2. Logic Micro-Operations

These are binary micro-operations carried out on the register bits. These procedures treat each bit as a binary variable and consider it separately.

Consider the X-OR micro-operation with the contents of R1 and R2 registers.

P: R1 ← R1 X-OR R2

A Control Function is also provided in the above statement.

3. Shift micro-operations

These are the important different types of micro-operations. That means we can move the register's contents to the left or right. The serial input shifts a bit to the rightmost position in the shift left operation, and a bit to the leftmost position in the shift right action.

There are three different sorts of shifts:

a) Logical Shift

The serial input is used to send 0 to the device. The symbols "shl" and "shr" are used to represent logical shifts left and right, respectively.

R1 ← she R1

R1 ← she R1

b) Circular Shift

This moves the bits of the register around the two ends without losing any data or contents. The shift register's serial output is connected to its serial input in this configuration. The terms "cir" and "cil" stand for left and right circular shifts, respectively.

d) Shift in Arithmetic

A signed binary number is shifted to the left or right using this method. Arithmetic shift left multiplies and divides a signed binary number by two. Because the signed number remains the same when multiplied or divided by two, the sign bit is left unaltered by the arithmetic shift micro-operation.

Conclusion

RTL is a symbolic notation for describing the micro-operations transfer between registers. It's a type of intermediate representation (IR) that's extremely similar to assembly language, such as the kind used in compilers. The term "Register Transfer" refers to the ability to perform micro-operations and then transfer the results to the same or another register.

Q.4 Explain Shift Micro-operations and Arithmetic Logic Shift Unit

Ans:- **Shift Micro-operations and Arithmetic Logic Shift Unit**

Shift Micro-operations (Hardware Implementation): a. Explain the concept of shift micro-operations. Provide examples of left and right shifts. b. Design a hardware implementation for a 4-bit shift register and demonstrate its operation with a specific input.

Arithmetic Logic Shift Unit (ALSU): a. Define Arithmetic Logic Shift Unit (ALSU) and explain its role in arithmetic and logical operations. b. Design a simple ALSU circuit using RTL and describe its operation with examples of arithmetic, logical, and shift operations.

Shift Micro-operations:

1. Introduction:

- **Definition:** Shift micro-operations involve the movement of binary digits either to the left or right within a register.
- **Purpose:** Used for manipulating data, facilitating multiplication or division by powers of two, and preparing data for logical or arithmetic operations.

2. Types of Shifts:

a. **Logical Shifts:** - Bits are moved without regard to their meaning. - **Left Shift (LSL):** All bits move to the left, and the vacated bit on the right is filled with a zero. - **Right Shift (LSR):** All bits move to the right, and the vacated bit on the left is filled with a zero.
b. **Arithmetic Shifts:** - Preserves the sign bit (the leftmost bit). - **Arithmetic Right Shift (ASR):** The sign bit is shifted into the empty position on the left.
c. **Circular Shifts:** - Bits that are shifted out at one end re-enter at the other end. - **Circular Left Shift (ROL):** Bits are rotated to the left. - **Circular Right Shift (ROR):** Bits are rotated to the right.

3. Hardware Implementation:

- **Shift Register:** Sequential logic circuit with flip-flops to store data.
- **Control Logic:** Determines the type and direction of shift.

4. Applications:

- **Data Encryption:** Shift operations are used in cryptographic algorithms.
- **Graphics Processing:** Shifts are employed for pixel manipulation and image transformation.
- **Signal Processing:** Useful in digital signal processing for filtering and convolution.

Arithmetic Logic Shift Unit (ALSU):

1. Definition:

- The Arithmetic Logic Shift Unit (ALSU) is a component of the CPU responsible for performing arithmetic operations, logical operations, and shift operations.

2. Components of ALSU:

a. **Arithmetic Unit:** - Performs addition, subtraction, multiplication, and division. b. **Logic Unit:** - Executes logical operations such as AND, OR, XOR, and NOT. c. **Shift Unit:** - Facilitates shift micro-operations.

3. Operations:

a. **Arithmetic Operations:** - Addition and subtraction are executed by the arithmetic unit. b. **Logical Operations:** - Logical AND, OR, XOR, and NOT are performed by the logic unit. c. **Shift Operations:** - Left and right shifts are carried out by the shift unit.

4. Control Unit:

- Coordinates the operations of the ALSU based on the instructions provided by the CPU.

5. Significance:

- The ALSU is crucial for the execution of arithmetic and logical instructions in a computer's central processing unit.
- Enhances the computational capabilities of the CPU by integrating diverse operations into a single unit.

6. Example:

- Consider a scenario where the ALSU is used to perform the operation (A + B) * C, where A, B, and C are 8-bit binary numbers.

Unit –III

Q .1: Instruction Codes and Computer Registers

1. **Stored Program Organization and Indirect Addressing:**

   a. **Stored Program Organization:**
   - *Definition:* Stored Program Organization refers to the architectural design where instructions and data are stored in the same memory unit. The program counter points to the memory location of the next instruction to be fetched and executed.
   - *Advantages:*
     - Flexibility: Enables the use of a wide range of instructions.
     - Efficient Execution: Allows for the sequential execution of instructions.

   b. **Indirect Addressing:**
   - *Definition:* Indirect addressing involves the use of an address held in a register or memory location to access the actual operand address.
   - *Example:* Consider an instruction **LOAD R1, (A)**, where the content of memory location A is loaded into register R1. Here, the indirect addressing mode is used to access the operand stored in memory location A.
   - *Advantages:*
     - Enhanced Flexibility: Enables dynamic changes in memory addressing.
     - Simplifies Program Logic: Reduces the need for multiple instructions to manipulate addresses.

2. **Common Bus Register:**

   a. **Common Bus Register Architecture:**
   - In a common bus register architecture, multiple registers are connected to a common data bus. Each register can read or write data to the bus.

   b. **Data Transfer:**
   - Data transfer between registers occurs by placing the data on the common bus. The source register places data on the bus, and the destination register reads it.

   c. **Advantages and Limitations:**
   - *Advantages:*
     - Simplicity: Simplifies the design of the data path.
     - Flexibility: Allows for easy communication between different parts of the CPU.
   - *Limitations:*
     - Limited Bandwidth: Potential contention for the common bus can limit data transfer speed.
     - Complexity in Large Systems: As the system scales, managing the common bus becomes more complex.

Q. 2: Computer Instructions and Timing Control

1.Set Completeness:

a. **Definition:**
- Instruction Set Completeness refers to having a sufficient set of instructions in a computer architecture to enable the implementation of any algorithm.

b. **Importance:**
- A complete instruction set provides the versatility needed to execute a wide range of programs without the need for additional instructions.

c. **Example:**
- Consider an instruction set that includes arithmetic (ADD, SUB), logical (AND, OR), and data transfer (MOV) instructions. With these instructions, it is possible to implement algorithms for arithmetic operations, logical comparisons, and data movement, demonstrating completeness.

2.Timing and Control:

a. **Role of Timing and Control:**
- Timing and control are critical for coordinating the execution of instructions in a computer.

b. **Control Unit and Timing Management:**
- The control unit manages the timing of different phases, such as fetch, decode, execute, and write back, ensuring the proper sequencing of operations.

c. **Significance:**
- Precise timing ensures that instructions are executed in the correct order, preventing data hazards and ensuring the reliability and performance of the computer system.

Q. 3: Instruction Cycle and Microprogrammed Control

1.Fetch and Decode in the Instruction Cycle:

a. **Steps in Fetch and Decode:**
- Fetch Phase: The program counter retrieves the instruction from memory, and the instruction is stored in the instruction register.
- Decode Phase: The control unit interprets the opcode in the instruction, determining the operation to be performed.

b. **Role of Program Counter and Instruction Register:**
- Program Counter: Points to the memory location of the next instruction.
- Instruction Register: Temporarily holds the fetched instruction for decoding.

c. **Example:**
- Consider the instruction **ADD R1, R2, R3**. During the fetch phase, the opcode is fetched from memory, and in the decode phase, the control unit identifies the instruction as an addition operation between registers R2 and R3, storing the result in register R1.

2.Types of Instructions and Register-Reference Instructions:

a. **R-type vs. I-type Instructions:**
- R-type instructions operate on data stored in registers, e.g., arithmetic operations.
- I-type instructions involve immediate values, e.g., loading a constant into a register.

b. **Register-Reference Instructions:**
- These instructions refer to operations involving registers directly, without immediate values.

c. **Examples:**
- R-type: **ADD R1, R2, R3** (add contents of R2 and R3, store in R1).
- I-type: **LOAD R1, #10** (load the immediate value 10 into register R1).

Q. 4: Control Memory and Addressing Sequencing

1.Microprogrammed Control:

a. **Definition:**
- Microprogrammed control involves using a set of microinstructions to control the operations of the control unit, allowing for a more flexible and easily modifiable control strategy.

b. **Advantages and Disadvantages:**
- *Advantages:*
  - Flexibility: Easily adaptable to changes in the instruction set.
  - Simplified Design: Microinstructions can be more straightforward than hardwired control circuits.
- *Disadvantages:*
  - Slower Execution: Microinstructions introduce additional steps, potentially slowing down instruction execution.
  - Complexity: Creating and managing microinstructions can be complex.

c. **Example:**
- Consider the microinstruction **0101 1100** that activates specific control signals for an addition operation. This microinstruction can be part of a set controlling various operations.

2.Addressing Sequencing (Conditional Branching, Mapping of Instructions, Subroutine):

a. **Conditional Branching:**
- Conditional branching involves altering the program flow based on a condition. The decision to branch is often based on flags set during execution.

b. **Mapping of Instructions:**
- Instructions are mapped to memory addresses, allowing the control unit to fetch and execute them sequentially.

c. **Subroutine:**
- Subroutine calls and returns involve transferring control to a different part of the program and then returning to the original execution point.

d. **Example:**

- Consider the instruction **BEQ address** (branch to the specified address if the zero flag is set). This illustrates conditional branching.

Unit IV

Q 1: Central Processing Unit (CPU) Introduction

1. **Introduction to Central Processing Unit (CPU):** a. **Definition and Role:**
   - The Central Processing Unit (CPU) is the primary component of a computer responsible for executing instructions. It performs arithmetic and logic operations, manages data storage and retrieval, and controls the overall operation of the computer.

   b. **Major Components and Functions:**
   - Components:
     - ALU (Arithmetic Logic Unit): Executes arithmetic and logic operations.
     - Control Unit: Manages the execution of instructions and controls data flow.
     - Registers: Temporary storage for data and addresses.
   - Functions:
     - Fetching instructions from memory.
     - Decoding instructions to determine operations.
     - Executing operations using the ALU.
     - Storing results in registers or memory.

   c. **CPU and System Interaction:**
   - The CPU communicates with other components, such as memory and I/O devices, through buses. It fetches and stores data in memory, interacts with input/output devices, and coordinates the overall functioning of the system.

Q. 2: General Register Organization and Micro-operations

1.General Register Organization:

a. **Concept:**
- General register organization involves using registers as storage elements in the CPU. Registers are small, fast storage locations within the CPU that temporarily hold data.

b. **Control Words:**
- Control words are instructions or commands that manage the operations of registers. They specify actions such as reading from or writing to registers, incrementing, or clearing register contents.

c. **Examples of Micro-operations:**
- Micro-operations are elementary operations that manipulate data within registers. Examples include:
  - **Transfer:** Move data from one register to another.
  - **Increment:** Increase the value of a register by one.
  - **Decrement:** Decrease the value of a register by one.
  - **Clear:** Set the content of a register to zero.

2.Examples of Micro-operations:

a. **Micro-operations Definition:**

- Micro-operations are elementary operations performed on data stored in registers. They include basic operations like transfer, arithmetic, and logic operations.

b. **Specific Examples:**

- **Transfer:** MOV R1, R2 (Transfer the content of register R2 to register R1).
- **Increment:** INC R3 (Increase the content of register R3 by 1).
- **Decrement:** DEC R4 (Decrease the content of register R4 by 1).
- **Clear:** CLR R5 (Set the content of register R5 to zero).

c. **Complex Tasks with Micro-operations:**

- Micro-operations can be combined to perform complex tasks. For example, loading data from memory into a register involves a sequence of transfer micro-operations.

Q. 3: Stack Organization and Evaluation of Arithmetic Expressions

1.Stack Organization:

a. **Differentiation:**

- *Register Stack:* Uses a set of registers as a stack for data storage.
- *Memory Stack:* Utilizes a portion of memory as a stack.

b. **Stack Concept:**

- A stack is a Last In, First Out (LIFO) data structure where data is pushed onto and popped off the top. It is commonly used for managing subroutine calls, expression evaluation, and other operations.

c. **Advantages and Disadvantages:**

- *Advantages:*
  - Simplifies program flow control.
  - Facilitates recursion and subroutine calls.
- *Disadvantages:*
  - Limited storage capacity.
  - Slower access compared to registers.

2.Reverse Polish Notation and Evaluation of Arithmetic Expressions: a. **RPN Definition:**

- Reverse Polish Notation (RPN) is a mathematical notation where operators are placed after their operands. It eliminates the need for parentheses in expressions.

b. **RPN in CPU with Stack:**

- CPU with stack organization evaluates RPN expressions by pushing operands onto the stack and performing operations when operators are encountered. The result is left on the stack.

c. **Example Evaluation:**

- Consider the RPN expression: **3 4 + 5 ***. The CPU would push 3 and 4 onto the stack, then encounter '+', performing the addition. The result (7) is pushed back. The CPU then encounters '5' and '*', resulting in the final value of 35 on the stack.

Q. 4: Instruction Formats

1.Three-Address, Two-Address, and One-Address Instruction Formats:

a. **Three-Address Instruction Format:**

- In three-address instructions, each instruction specifies three operands, where the result is stored. Example: **ADD R1, R2, R3** (R1 = R2 + R3).

b. **Two-Address Instruction Format:**

- In two-address instructions, one operand serves as both a source and destination. Example: **SUB R1, R2** (R1 = R1 - R2).

c. **One-Address Instruction Format:**

- In one-address instructions, operations are performed between a register and an implied accumulator. Example: **INC R1** (R1 = R1 + 1).

2.Advantages and Disadvantages of Different Instruction Formats: a. **Comparison:**

- *Three-Address:* Supports complex operations but may require more memory.
- *Two-Address:* Efficient use of memory but limits expressiveness.
- *One-Address:* Simple, but may require extra instructions for complex operations.

b. **Impact on CPU Design:**

- The choice of instruction format influences the complexity of the control unit and the overall design of the CPU.

c. **Real-world Examples:**

- Complex scientific computations may benefit from three-address instructions, while embedded systems may favor one or two-address instructions for simplicity and efficiency.

Unit – V

Q. 1: Central Processing Unit - Addressing Modes

1. **Addressing Modes:** a. **Definition:**
   - Addressing modes refer to the various ways in which a CPU can specify the operands of an instruction, i.e., the location from which data is to be fetched or to which it is to be stored.

   b. **Significance of Addressing Modes:**
   - Different addressing modes offer flexibility in programming and optimization.
   - Immediate addressing: Operand is specified within the instruction.
   - Direct addressing: Operand address is directly given in the instruction.
   - Indirect addressing: Address of the operand is stored in a register or memory location.
   - Register addressing: Operand is in a register.

   c. **Examples:**
   - Immediate: **MOV R1, #5** (Move the immediate value 5 into register R1).
   - Direct: **LOAD R1, A** (Load the content of memory location A into register R1).
   - Indirect: **LOAD R1, (A)** (Load the content of the memory location whose address is stored in A into register R1).
   - Register: **ADD R1, R2** (Add the content of registers R1 and R2).

Q. 2: Data Transfer and Manipulation

1.Data Transfer and Manipulation:

a. **Differentiation:**
- *Data Transfer Instructions:* Involve moving data between registers and memory.
- *Data Manipulation Instructions:* Include operations that modify the content of registers.

b. **Examples of Data Transfer Instructions:**
- **MOV R1, [A]** (Move the content of memory location A into register R1).
- **STORE R1, [B]** (Store the content of register R1 into memory location B).

c. **Arithmetic, Logical, and Bit Manipulation Instructions:**
- *Arithmetic Instructions:* Perform mathematical operations on data in registers.
  - **ADD R1, R2** (R1 = R1 + R2).
- *Logical Instructions:* Perform logical operations (AND, OR, XOR) on data.
  - **AND R1, R2** (R1 = R1 AND R2).
- *Bit Manipulation Instructions:* Modify individual bits in registers.
  - **SET_BIT R1, 3** (Set the 3rd bit of register R1).

2.Shift Instructions:

a. **Definition:**
- Shift instructions involve moving the bits of a binary number left or right.

b. **Types of Shifts:**
- *Logical Shifts:* Empty bit positions are filled with zeros.
- *Arithmetic Shifts:* Preserve the sign bit, filling empty positions with the sign bit.

c. **Examples:**
- **LSL R1, 2** (Left shift the content of register R1 by 2 positions).
- **ASR R2, 1** (Arithmetic right shift the content of register R2 by 1 position).

Q. 3: Program Control

1.Status Bit Conditions:

a. **Concept of Status Bit Conditions:**
- Status bit conditions are flags that represent the outcome of an operation.
- Common flags include the zero flag (Z), carry flag (C), and overflow flag (V).

b. **Common Status Bits:**
- *Zero Flag (Z):* Set if the result of an operation is zero.
- *Carry Flag (C):* Set if there is a carry-out or borrow during an operation.
- *Overflow Flag (V):* Set if the result of a signed operation exceeds the representable range.

c. **Influence on Program Flow:**
- Status bit conditions influence conditional branches and decisions in the program flow.
- Example: **BEQ label** (Branch to label if the zero flag is set).

2.Conditional Branch Instructions:

a. **Definition of Conditional Branch Instructions:**
- Conditional branch instructions alter the program counter based on specific conditions.

b. **Examples:**
- **BEQ label** (Branch to label if the zero flag is set).
- **BGT label** (Branch to label if the result is greater than zero).

c. **Advantages of Conditional Branches:**
- Enable the implementation of decision-making structures.
- Improve program efficiency by skipping unnecessary instructions.

Q. 4: Program Control - Interrupts

1.Program Interrupt:

a. **Definition:**

- A program interrupt is an event that causes the CPU to temporarily stop executing the current program and transfer control to another part of the system.

b. **Differentiation:**

- *Software Interrupt:* Triggered by an instruction in the program.
- *Hardware Interrupt:* External events such as I/O requests or timer expiration.

c. **CPU Response to Interrupts:**

- The CPU responds by saving the current state, executing a specific routine (interrupt service routine or ISR), and later returning to the interrupted program.
-

2.Types of Interrupt: a. **Types:**

- *External Hardware Interrupts:* Initiated by external devices, e.g., keyboard input.
- *Internal Software Interrupts:* Generated by software, e.g., system calls.

b. **Maskable and Non-Maskable Interrupts:**

- *Maskable Interrupts:* Can be temporarily disabled by the CPU.
- *Non-Maskable Interrupts:* Always take precedence and cannot be disabled.

c. **Examples:**

- *External:* Timer interrupt from a peripheral device.
- *Internal:* Division by zero exception.