Unit -1

1) What is Register Transfer Language (RTL)? And register transfer.

2) What is Micro-operation? Explain Shift, Arithmetic and Logical Shift Micro-operation?

3) Explain Basic Computer Instruction Formats?

(Memory reference instruction, register reference instruction and I/O reference instruction)

### 1. Register Transfer Language (RTL) and Register Transfer

\*\*Register Transfer Language (RTL):\*\*

RTL is a symbolic representation used in computer engineering to describe the operations, data transfers, and transformations that occur between registers within a digital system. It provides a precise way to specify the low-level operations performed during the execution of instructions by a CPU. RTL expressions are used to define how data moves between registers, memory, and other components, often as part of the instruction cycle.

\*\*Register Transfer:\*\*

Register transfer refers to the movement of data from one register to another within a digital system. This is an essential operation in the execution of machine instructions, where data is read from source registers, processed by the ALU (Arithmetic Logic Unit) or other functional units, and then written back to destination registers.

### 2. Micro-operation and Shift Operations

\*\*Micro-operation:\*\*

A micro-operation is a fundamental operation that occurs at the level of registers in a computer's CPU. These operations typically involve moving data between registers, performing arithmetic or logical operations on register contents, or shifting data within registers. Micro-operations are the building blocks of more complex machine instructions and are executed during specific phases of the instruction cycle.

\*\*Shift Micro-operations:\*\*

Shift micro-operations move the bits of a register's contents to the left or right. There are three types of shifts:

- \*\*Logical Shift:\*\*

- \*\*Logical Left Shift (SLL):\*\* Moves bits to the left, with zeros shifted into the least significant bit (LSB). For example, shifting 1010 left by one bit results in 0100.

- \*\*Logical Right Shift (SRL):\*\* Moves bits to the right, with zeros shifted into the most significant bit (MSB). For example, shifting 1010 right by one bit results in 0101.

- \*\*Arithmetic Shift:\*\*

- \*\*Arithmetic Left Shift:\*\* Similar to the logical left shift, but it is typically used in arithmetic operations.

- \*\*Arithmetic Right Shift:\*\* Moves bits to the right, with the sign bit (MSB) retained to preserve the number's sign in signed binary numbers. For example, shifting 1101 (signed -3) right by one bit results in 1110 (signed -2).

- \*\*Circular Shift (Rotate):\*\*

- \*\*Rotate Left (ROL):\*\* Moves bits to the left, with the MSB wrapping around to the LSB. For example, rotating 1010 left by one bit results in 0101.

- \*\*Rotate Right (ROR):\*\* Moves bits to the right, with the LSB wrapping around to the MSB. For example, rotating 1010 right by one bit results in 0101.

### 3. Basic Computer Instruction Formats

In a basic computer, instructions can typically be categorized into three main formats based on their reference type: memory reference instructions, register reference instructions, and I/O reference instructions.

\*\*Memory Reference Instructions:\*\*

These instructions involve operations that require accessing memory locations. They usually have the following format:

```

Opcode (7 bits) | Address (12 bits)

```

The opcode specifies the operation to be performed, and the address specifies the memory location involved. Examples include LOAD, STORE, and ADD operations.

\*\*Register Reference Instructions:\*\*

These instructions involve operations that are performed directly on the CPU registers without accessing memory. They typically have the following format:

```

Opcode (7 bits) | Register (12 bits)

```

Here, the opcode specifies the operation, and the register field specifies which CPU register is involved. Examples include CLEAR, INCREMENT, and COMPLEMENT operations.

\*\*I/O Reference Instructions:\*\*

These instructions are used for input and output operations involving peripheral devices. They usually have the following format:

```

Opcode (7 bits) | I/O Address or Command (12 bits)

```

The opcode specifies the I/O operation, and the address or command field specifies the particular I/O device or operation. Examples include INPUT, OUTPUT, and various control instructions for managing I/O operations.

In summary, these instruction formats define how the CPU interprets and executes various types of operations by specifying the necessary opcodes and operands, whether they involve memory, registers, or I/O devices.

Unit -2

1) What is machine Language and Assembly Language? Also differentiate between them.

2) Explain Address Sequence with Diagram.

3) Explain design of control unit with Diagram.

4) Short Questions

I. Control Memory

II. Assembler

III. Subroutines

IV. Program Loops

### 1) Machine Language and Assembly Language:

\*\*Machine Language\*\*:

- The lowest-level programming language.

- Consists of binary code (0s and 1s) that the computer's central processing unit (CPU) can execute directly.

- Highly specific to the architecture of the CPU.

- Difficult for humans to read and write due to its complexity and lack of abstraction.

\*\*Assembly Language\*\*:

- A low-level programming language that is one step above machine language.

- Uses mnemonics (symbolic names) instead of binary codes, making it easier to read and write.

- Each assembly language instruction corresponds directly to a machine language instruction.

- Requires an assembler to translate assembly code into machine code.

- Architecture-specific, meaning it differs between different types of CPUs.

\*\*Differences\*\*:

1. \*\*Representation\*\*:

- Machine Language: Binary code (e.g., 10101010).

- Assembly Language: Mnemonics (e.g., MOV AX, BX).

2. \*\*Human Readability\*\*:

- Machine Language: Not human-readable.

- Assembly Language: More human-readable with symbolic names.

3. \*\*Translation\*\*:

- Machine Language: Directly executed by the CPU.

- Assembly Language: Requires an assembler to convert into machine language.

### 2) Address Sequence with Diagram:

The address sequence in computer architecture typically refers to the order in which memory addresses are accessed during the execution of a program. This sequence is crucial for understanding how a program moves through instructions and accesses data.

#### Diagram:

```

+------------+----------------+----------------+--------------+

| Address | Instruction | Next Address | Description |

+------------+----------------+----------------+--------------+

| 0x0000 | Load A, 0x0005 | 0x0001 | Load data from address 0x0005 into register A |

| 0x0001 | Add A, 0x0006 | 0x0002 | Add data from address 0x0006 to register A |

| 0x0002 | Store A, 0x0007| 0x0003 | Store data from register A to address 0x0007 |

| 0x0003 | Jump 0x0000 | 0x0000 | Jump back to address 0x0000 |

+------------+----------------+----------------+--------------+

```

This simple example demonstrates a sequence of addresses and their corresponding instructions in a program loop.

### 3) Design of Control Unit with Diagram:

The control unit (CU) is a component of a computer's CPU that directs the operation of the processor. It tells the computer's memory, arithmetic and logic unit (ALU), and input/output devices how to respond to the instructions that have been sent to the processor.

#### Diagram:

```

+--------------------+

| |

| Control Unit |

| |

+--------------------+

|

+---------------+---------------+

| |

+-------v--------+ +--------v-------+

| Instruction | | Control |

| Register (IR) | | Logic |

| | | Circuits |

+----------------+ +--------+--------+

|

+----------------+----------------+

| | |

| | |

+--------v--------+ +-----v------+ +-------v------+

| Timing & | | Control | | Status |

| Control | | Signals | | Flags |

| Generator | +-------------+ +--------------+

+-----------------+

```

\*\*Components\*\*:

1. \*\*Instruction Register (IR)\*\*: Holds the instruction currently being executed.

2. \*\*Control Logic Circuits\*\*: Generate control signals based on the contents of the IR.

3. \*\*Timing and Control Generator\*\*: Ensures operations occur in the correct sequence and timing.

4. \*\*Control Signals\*\*: Direct the operation of the CPU and peripherals.

5. \*\*Status Flags\*\*: Indicate the state of the processor (e.g., zero flag, carry flag).

### 4) Short Questions:

I. \*\*Control Memory\*\*:

- Special memory used by the control unit of a CPU.

- Contains microprograms that define the operations of the control unit.

- Helps in implementing complex control logic in a structured way.

II. \*\*Assembler\*\*:

- A software tool that translates assembly language programs into machine language.

- Converts mnemonic code into binary instructions.

- Resolves symbolic names for addresses and constants.

III. \*\*Subroutines\*\*:

- Blocks of code designed to perform a specific task, reusable throughout a program.

- Called from different parts of the program.

- Helps in code modularity and reduces redundancy.

- Typically involves saving the state of the current execution (using the stack) before jumping to the subroutine and restoring it after completion.

IV. \*\*Program Loops\*\*:

- Structures in programming that repeat a sequence of instructions until a certain condition is met.

- Types include "for loops", "while loops", and "do-while loops".

- Used for tasks that require repetitive operations, like iterating over arrays or generating sequences.

```text

Example of a simple program loop in pseudo-code:

FOR i = 1 TO 10

PRINT i

END FOR

```

This loop prints numbers from 1 to 10 by iterating 10 times.

Unit -3

1) What is Addressing Modes? Explain different types of address mode with ex.

2) Explain and Draw the flowchart of Instruction Cycle.

3) What is vector processing and array processor?

4) Explain different types of Flynn’s Classification.

5) What is pipelining? Explain Arithmetic pipeline?

6) What is parallel processing?

Let's address each of your queries one by one:

### 1) Addressing Modes

Addressing modes are the methods used in computer instructions to specify the location of an operand (data). These modes allow the CPU to access operands in different ways, enhancing flexibility in instruction execution.

#### Types of Addressing Modes:

1. \*\*Immediate Addressing Mode\*\*:

- Operand is specified in the instruction itself.

- Example: `MOV A, #5` (Here, 5 is the immediate operand)

2. \*\*Register Addressing Mode\*\*:

- Operand is in a register.

- Example: `MOV A, B` (Here, B is a register containing the operand)

3. \*\*Direct Addressing Mode\*\*:

- Operand’s address is given explicitly in the instruction.

- Example: `MOV A, 1000` (Here, 1000 is the memory address where the operand is located)

4. \*\*Indirect Addressing Mode\*\*:

- Instruction specifies a memory location that holds the address of the operand.

- Example: `MOV A, @B` (Here, the memory location whose address is in B contains the operand)

5. \*\*Indexed Addressing Mode\*\*:

- Operand’s address is determined by adding a constant value to the contents of a register.

- Example: `MOV A, 1000(BX)` (Here, the effective address is 1000 + the value in BX)

6. \*\*Base Register Addressing Mode\*\*:

- Similar to indexed but the base address is held in a register.

- Example: `MOV A, (BX)` (Here, the effective address is the value in BX)

7. \*\*Relative Addressing Mode\*\*:

- Used in branch instructions; operand is at an address relative to the current instruction’s address.

- Example: `JMP 100` (Here, jump to the address current address + 100)

### 2) Instruction Cycle

The instruction cycle is the process by which a computer retrieves, decodes, and executes instructions from its memory.

#### Flowchart:

1. \*\*Fetch Cycle\*\*:

- Fetch the instruction from memory.

- Increment the program counter (PC).

2. \*\*Decode Cycle\*\*:

- Decode the fetched instruction to determine the operation and the operands.

3. \*\*Execute Cycle\*\*:

- Execute the decoded instruction using the ALU, registers, and memory.

4. \*\*Store Cycle\*\* (if necessary):

- Store the result back into memory or a register.

#### Flowchart:

```

Start

|

Fetch Instruction

|

Increment PC

|

Decode Instruction

|

Execute Instruction

|

Store Result (if needed)

|

Check for Interrupt

|

|-------> Interrupt Handling (if any)

| |

Yes No

| |

Resume End

```

### 3) Vector Processing and Array Processor

- \*\*Vector Processing\*\*:

- Vector processing involves the computation of vector operands in a single instruction. It’s suited for scientific calculations and tasks requiring large data arrays.

- Example: Operations on matrices.

- \*\*Array Processor\*\*:

- An array processor is a specialized computer designed to process large arrays of data in parallel. It uses multiple processing elements to perform the same operation on different data simultaneously.

- Example: Used in applications like image processing and simulations.

### 4) Flynn’s Classification

Flynn's Taxonomy is a classification of computer architectures based on the number of concurrent instruction and data streams:

1. \*\*SISD (Single Instruction stream, Single Data stream)\*\*:

- A single processor executes a single instruction stream to operate on data stored in a single memory.

- Example: Traditional uniprocessor.

2. \*\*SIMD (Single Instruction stream, Multiple Data streams)\*\*:

- A single instruction controls multiple processing elements to perform the same operation on multiple data points simultaneously.

- Example: Vector processors.

3. \*\*MISD (Multiple Instruction streams, Single Data stream)\*\*:

- Multiple instructions operate on a single data stream. This architecture is not widely used.

- Example: Fault-tolerant computers.

4. \*\*MIMD (Multiple Instruction streams, Multiple Data streams)\*\*:

- Multiple processors independently execute different instructions on different data.

- Example: Modern multicore processors.

### 5) Pipelining and Arithmetic Pipeline

- \*\*Pipelining\*\*:

- Pipelining is a technique where multiple instruction phases are overlapped. It’s akin to an assembly line, where different stages of instruction execution are handled simultaneously.

- \*\*Arithmetic Pipeline\*\*:

- An arithmetic pipeline breaks down arithmetic operations into sub-operations that can be performed concurrently in different pipeline stages.

- Example: Floating-point multiplication can be split into several stages like fetching operands, multiplying mantissas, adding exponents, and normalizing the result.

### 6) Parallel Processing

Parallel processing involves the simultaneous use of multiple processors or processor cores to perform computations more quickly. It divides a problem into sub-problems, solves them concurrently, and then combines the results.

- \*\*Example\*\*:

- Large-scale scientific computations, real-time data processing, and high-performance computing applications all benefit from parallel processing.

Unit -4

1) Explain Booth Multiplication Algorithm.

Solve (-23) \*(-19).

2) What is DMA? Explain DMA Transfer and DMA Controller with diagram?

3) Explain IOP with diagram?

4) What is Asynchronous Data Transfer?

Let's address your queries one by one:

### 1) Booth Multiplication Algorithm

Booth's Multiplication Algorithm is an efficient algorithm for multiplying binary numbers in two's complement notation. It reduces the number of addition operations required when performing multiplication.

#### Steps of Booth's Algorithm:

1. \*\*Initialize\*\*:

- Let \( M \) be the multiplicand and \( Q \) be the multiplier.

- Initialize \( A \) (Accumulator) to 0.

- Extend \( Q \) by one bit to the right (initially set to 0), called \( Q\_{-1} \).

- Set a counter equal to the number of bits in \( Q \).

2. \*\*Perform the following steps until the counter reaches 0\*\*:

- Check the two least significant bits of \( Q \) and \( Q\_{-1} \).

- \*\*If \( Q\_0Q\_{-1} = 01 \)\*\*: Subtract \( M \) from \( A \) (i.e., \( A = A - M \)).

- \*\*If \( Q\_0Q\_{-1} = 10 \)\*\*: Add \( M \) to \( A \) (i.e., \( A = A + M \)).

- \*\*If \( Q\_0Q\_{-1} = 00 \) or \( 11 \)\*\*: No operation.

- Perform an arithmetic right shift on the concatenated \( AQ \) (right shift both \( A \) and \( Q \) and move the previous \( Q\_0 \) to \( Q\_{-1} \)).

3. \*\*End\*\*: After all iterations, \( A \) and \( Q \) together represent the product of the multiplication.

#### Example: Multiplying (-23) and (-19)

- Represent -23 and -19 in binary (8-bit representation):

-23: 11101001

-19: 11101101

Let's use Booth's algorithm:

1. Initialize:

- \( M = -23 = 11101001 \)

- \( Q = -19 = 11101101 \)

- \( Q\_{-1} = 0 \)

- \( A = 0 \)

- Counter = 8 (number of bits)

2. Perform the steps:

| Step | \( A \) | \( Q \) | \( Q\_{-1} \) | Action |

|------|---------------|---------------|--------------|---------------|

| 0 | 00000000 | 11101101 | 0 | Initial |

| 1 | 00000000 | 11101101 | 0 | No operation |

| 2 | 11101001 | 11101101 | 0 | Add \( M \) |

| 3 | 11110100 | 11110110 | 1 | Right shift |

| 4 | 00000011 | 11110110 | 1 | Subtract \( M \)|

| 5 | 00000001 | 11111011 | 0 | Right shift |

| 6 | 11101001 | 11111011 | 0 | Add \( M \) |

| 7 | 11110100 | 11111101 | 1 | Right shift |

| 8 | 00000011 | 11111101 | 1 | Subtract \( M \)|

| 9 | 00000001 | 11111110 | 0 | Right shift |

After 8 steps, the result stored in \( AQ \) is 0000000111111110 which equals 437 in decimal. The product of -23 and -19 is indeed 437, verifying that Booth's algorithm is correctly implemented.

### 2) Direct Memory Access (DMA)

DMA is a feature that allows certain hardware subsystems within a computer to access the main system memory (RAM) independently of the central processing unit (CPU).

#### DMA Transfer:

DMA transfers data directly between I/O devices and memory, bypassing the CPU to free it up for other tasks.

#### DMA Controller:

The DMA controller is a hardware component that manages the DMA transfers. It takes control of the system bus for data transfer operations.

#### DMA Transfer Process:

1. The CPU initializes the DMA controller by providing it with the necessary information (memory address, I/O address, the number of bytes to transfer, and the direction of transfer).

2. The DMA controller then takes control of the system bus to transfer data directly between the memory and the I/O device.

3. Once the transfer is complete, the DMA controller signals the CPU by sending an interrupt.

#### DMA Controller Diagram:

```

+----------------------+

| CPU |

+----------+-----------+

|

| Address and Control Bus

|

+----------+-----------+

| DMA Controller |

+----------+-----------+

|

| Data Bus

|

+-------+------+ +--------+-------+

| Memory | | I/O Device |

+---------------+ +---------------+

```

### 3) Input/Output Processor (IOP)

An IOP is a specialized processor designed to handle I/O tasks. It manages data transfer between the main processor and peripheral devices, offloading these tasks from the main CPU to improve system performance.

#### IOP Operation Diagram:

```

+------------------------+

| Main CPU |

+-----------+------------+

|

| Control/Status

|

+-----------+------------+

| IOP |

+-----------+------------+

|

| Data Bus

|

+-----------+------------+

| Memory | I/O Devices|

+-----------+-------------+

```

### 4) Asynchronous Data Transfer

Asynchronous data transfer refers to the transfer of data without a synchronized clock signal. The sender and receiver do not share a common timing signal, and data is transferred using handshaking signals to ensure proper timing.

#### Characteristics:

- \*\*No Common Clock\*\*: The sender and receiver operate independently.

- \*\*Handshaking\*\*: Utilizes control signals to coordinate the transfer.

- \*\*Flexible Timing\*\*: Suitable for varying data rates.

#### Handshaking Protocol:

1. \*\*Request\*\*: The sender asserts a request signal indicating data is ready to be sent.

2. \*\*Acknowledge\*\*: The receiver asserts an acknowledge signal when it is ready to receive the data.

3. \*\*Transfer\*\*: Data is transferred once the acknowledge signal is received.

4. \*\*Completion\*\*: The sender deasserts the request signal, and the receiver deasserts the acknowledge signal upon completion of the transfer.

This method ensures reliable data transfer without needing synchronized clocks, making it ideal for peripheral devices with variable data rates.

Unit -5

1) What is Cache coherence Problem.

2) What is multiprocessor? Explain the various characteristics of multiprocessor?

3) What is inter-processor communication?

4) What is Associative memory?.

5) What is virtual memory?

6) What is cache memory? Explain various mapping of cache memory?

Sure, let's go through these concepts one by one:

### 1) Cache Coherence Problem

The cache coherence problem occurs in multiprocessor systems where multiple processors have their own local caches. Each cache stores copies of data from the main memory. The problem arises when one processor updates data in its cache, but other caches have stale copies of that data. Ensuring that all caches have the most recent data and maintaining consistency across all caches is known as cache coherence. Common cache coherence protocols include:

- \*\*Write-invalidate\*\*: When a processor writes to a location, it invalidates the same location in other caches.

- \*\*Write-update (or write-broadcast)\*\*: When a processor writes to a location, it updates the value in all other caches.

### 2) Multiprocessor

A multiprocessor is a computer system with two or more central processing units (CPUs) that share a common physical memory. These systems can execute multiple processes simultaneously, enhancing performance and reliability.

#### Characteristics of Multiprocessors:

- \*\*Tightly Coupled System\*\*: Multiprocessors share a common memory space and are connected via a high-speed bus.

- \*\*Parallelism\*\*: Multiple processors work on different parts of a problem simultaneously.

- \*\*Scalability\*\*: Systems can be expanded by adding more processors.

- \*\*Synchronization\*\*: Requires mechanisms to ensure that processors coordinate properly and handle shared resources without conflicts.

- \*\*Load Balancing\*\*: Efficient distribution of tasks among processors to ensure optimal utilization of resources.

### 3) Inter-Processor Communication

Inter-processor communication refers to the methods and protocols used for exchanging data and signals between processors in a multiprocessor system. This can be achieved through:

- \*\*Shared Memory\*\*: Processors communicate by reading and writing to shared memory locations.

- \*\*Message Passing\*\*: Processors send and receive messages through an interconnection network, often used in distributed systems.

- \*\*Synchronization Primitives\*\*: Mechanisms like semaphores, mutexes, and barriers to manage access to shared resources and coordinate actions among processors.

### 4) Associative Memory

Associative memory, also known as content-addressable memory (CAM), allows data retrieval based on the content rather than a specific address. This memory type is highly efficient for applications requiring fast lookups, such as:

- \*\*Routing tables in networking devices\*\*.

- \*\*Cache memory in computers\*\*.

- \*\*Pattern recognition systems\*\*.

### 5) Virtual Memory

Virtual memory is a memory management technique that provides an "idealized abstraction" of the storage resources that are actually available on a given machine. It creates an illusion for users of a very large (main) memory:

- \*\*Paging\*\*: Divides the virtual memory into blocks of physical memory called pages.

- \*\*Segmentation\*\*: Divides the program's memory into different segments based on the logical divisions.

- \*\*Benefits\*\*: It allows for efficient and safe use of the memory, providing the ability to run larger applications on systems with less physical memory and enabling memory protection and isolation between processes.

### 6) Cache Memory

Cache memory is a small, high-speed storage located close to the CPU. It stores copies of frequently accessed data from the main memory, thus reducing the average time to access data.

#### Mapping Techniques of Cache Memory:

- \*\*Direct Mapping\*\*: Each block of main memory maps to only one possible cache line.

- \*\*Pros\*\*: Simple and inexpensive.

- \*\*Cons\*\*: Can lead to conflicts if multiple blocks map to the same cache line.

- \*\*Fully Associative Mapping\*\*: Any block can be placed in any cache line.

- \*\*Pros\*\*: Flexible and reduces conflict misses.

- \*\*Cons\*\*: Expensive and complex to implement due to the need to check all cache lines.

- \*\*Set-Associative Mapping\*\*: A combination of direct and fully associative mapping. The cache is divided into several sets, and each block maps to any line within a set.

- \*\*Pros\*\*: Balances complexity and performance.

- \*\*Cons\*\*: More complex than direct mapping, less flexible than fully associative mapping.

These mappings help determine how memory blocks are placed in the cache and how the cache handles retrieval and replacement of data.