QUESTION BANK ANSWERS

GTU

(1.) What do u mean by register transfer ? Explain in details.Also discuss three state buffer.

Answer:-

Register Transfer is a fundamental concept in digital systems and computer architecture. It refers to the process of moving data between registers (or storage locations) within a computer system. Here's a detailed explanation:

1. **Registers**: Registers are small, fast storage locations in a CPU used to hold data temporarily. They are crucial for executing instructions and performing calculations.

2. **Transfer of Data**: Register transfer involves moving data from one register to another or performing operations on the data stored in registers. This can include:

- **Loading** data from memory or an input device into a register.
- **Storing** data from a register into memory or an output device.
- **Transferring** data between two registers for further processing.

3. **Register Transfer Language (RTL)**: This is a notation used to describe the operations performed in register transfer. It specifies:

- **Source Register**: The register from which data is being transferred. -

Destination Register: The register to which data is being transferred. -

Operation: The action performed, such as loading, storing, or moving data.

For example, in RTL, a statement like `R1 <- R2` means "transfer the contents of register R2 to register R1."

4. **Control Signals**: The process is controlled by various signals within the CPU. These signals determine when and how data is transferred between registers.

5. **Purpose**: Register transfer operations are essential for implementing instructions in a CPU. They facilitate operations like arithmetic calculations, data movement, and control flow in a computer program.

Three-State Bus Buffer

A **Three-State Bus Buffer** (also known as a tri-state buffer) is a component used in digital circuits to manage data flow on a bus. It can be in one of three states:

1. **High (1)**: The buffer outputs a high voltage level (logic 1).

2. **Low (0)**: The buffer outputs a low voltage level (logic 0).

3. **High Impedance (Z)**: The buffer is in a high-impedance state, effectively disconnecting itself from the bus. This state is also known as the "off" state.

Here's a breakdown of the three-state bus buffer's operation and significance:

1. **Purpose**: The primary purpose of a three-state buffer is to allow multiple devices to share a common data bus without interfering with each other. Only one device drives the bus at a time, while others are in a high-impedance state, preventing conflicts.

2. **Control Input**: A control input, often called the enable or gate signal, determines whether the buffer is in a high, low, or high-impedance state. When the control signal is active (e.g., high), the buffer either drives the bus with a high or low value. When the control signal is inactive (e.g., low), the buffer goes into the high-impedance state.

3. **Bus Management**: In a bus system, multiple devices can be connected to the same set of lines (data bus). To ensure that only one device communicates at any given time, three-state buffers are used. They effectively isolate the bus from devices that are not currently communicating.

4. **Applications**: Three-state buffers are widely used in memory systems, I/O interfaces, and communication systems to manage data flow and prevent bus contention (where multiple devices attempt to drive the bus simultaneously).

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

Answers-

Types of Shift Operations on Accumulator

SUBJECT :- CA SUBJECT CODE-C2310C2

Shift operations are fundamental operations in computer architecture, used for various purposes like multiplication, division, bit manipulation, and data formatting. They involve moving the bits within a register (in this case, the accumulator) to the left or right.

Here are the primary types of shift operations:

1. Logical Shift

* **Logical Left Shift (LSL):** Shifts all bits in the accumulator one position to the left. The least significant bit (LSB) is shifted out and discarded, while the most significant bit (MSB) is replaced with a 0.

* **Logical Right Shift (LSR):** Shifts all bits in the accumulator one position to the right. The MSB is shifted out and discarded, while the LSB is replaced with a 0.

2. Arithmetic Shift

Used primarily for signed numbers.

* **Arithmetic Left Shift (ASL):** Similar to LSL, but the sign bit (MSB) is preserved. The LSB is replaced with a 0.

* **Arithmetic Right Shift (ASR):** Shifts all bits in the accumulator one position to the right. The LSB is shifted out and discarded, while the MSB (sign bit) is preserved.

3. Circular Shift (Rotate)

* **Circular Left Shift (ROL):** Shifts all bits in the accumulator one position to the left. The bit shifted out from the MSB is inserted into the LSB.

* **Circular Right Shift (ROR):** Shifts all bits in the accumulator one position to the right. The bit shifted out from the LSB is inserted into the MSB.

Key Points

* **Purpose:** Shift operations are used for multiplication, division, bit manipulation, and data formatting.

* **Accumulator:** The register where the shift operation takes place.

* **Bit Movement:** Bits are moved to the left or right within the accumulator.

* **Bit Replacement:** The vacated positions are filled with 0s (logical shift), the sign bit (arithmetic shift), or the shifted-out bit (circular shift).

Example:

Consider an accumulator with the value 1011 (decimal 11).

* LSL: 0110 (decimal 6)
* LSR: 0101 (decimal 5)
* ASL: 0110 (decimal 6)
* ASR: 1011 (decimal -5, assuming 2's complement representation)
* ROL: 0111 (decimal 7)
* ROR: 0010 (decimal 2)

(3.)Define RTL. Explain how register transfer takes place in basic computer system. Answers-

Register Transfer Language (RTL)

RTL is a symbolic notation used to describe the transfer of data between registers within a computer's processing unit. It's essentially a way to represent the micro-operations that occur at the hardware level.

Key components of RTL:

* **Registers:** High-speed storage elements that hold data

temporarily. * **Data paths:** Connections between registers for

transferring data.

* **Control signals:** Signals that determine the operation to be

performed. ### Register Transfer in a Basic Computer System

A basic computer system consists of several key components, including:

* **CPU (Central Processing Unit):** The brain of the computer, responsible for executing instructions.

* **Memory:** Stores data and instructions.

* **Input/Output devices:** Interact with the outside world.

SUBJECT :- CA SUBJECT CODE-C2310C2 **Register transfer** is the mechanism by which data moves

between these components.

How it works:

1. **Instruction Fetch:** The CPU fetches an instruction from memory and stores it in an instruction register.

2. **Instruction Decode:** The instruction is decoded to determine the operation to be performed and the registers involved.

3. **Data Transfer:** Data is moved between registers, memory, and input/output devices as required by the instruction. This involves activating control signals to open and close data paths between the desired components.

4. **Arithmetic/Logic Operation:** The CPU performs the specified arithmetic or logical operation on the data in registers.

5. **Data Storage:** The result of the operation is stored in a register or memory

location. **Example:**

Consider a simple instruction: `Add R1, R2`

* The CPU fetches the instruction and decodes it.

* The contents of registers R1 and R2 are transferred to the ALU (Arithmetic Logic

Unit). * The ALU performs the addition operation.

* The result is stored back in register R1.

Visual representation:

[Image of a basic computer architecture with data paths and control signals]

Key points to remember:

* Register transfer is the fundamental operation of a computer system.

* It is controlled by control signals generated by the control unit.

* Data paths connect various components of the system.

SUBJECT :- CA SUBJECT CODE-C2310C2

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

Answers

Multiplexing

Multiplexing is a process of combining multiple input signals into a single output signal for transmission over a shared medium. This technique is essential for efficient utilization of communication channels.

Multiplexing of Control Signals in ALU

An Arithmetic Logic Unit (ALU) performs various arithmetic and logical operations on data. To control these operations, multiple control signals are required. Multiplexing these control signals is crucial for reducing the number of control lines, simplifying the ALU's control logic, and improving overall performance.

How it works:

Control Signal Generation: Multiple control signals are generated based on the instruction being executed. These signals determine the specific operation to be performed by the ALU.

Multiplexer Implementation: A multiplexer is used to combine these multiple control signals into a single control bus. The select lines of the multiplexer determine which control signal is passed to the ALU.

ALU Operation: The selected control signal from the multiplexer activates the corresponding circuitry within the ALU to perform the desired operation.

Benefits of Multiplexing Control Signals:

Reduced number of control lines: By combining multiple control signals into a single bus, the number of control lines required is significantly reduced.

Simplified control logic: Multiplexing simplifies the control logic of the ALU as it requires fewer control signals to be generated and managed.

Improved performance: By reducing the number of control lines, the overall speed of the ALU can be improved.

Example:

Consider an ALU that can perform four operations: addition, subtraction, AND, and OR. Four control signals (A, B, C, D) can be generated to represent these operations. A 2-to-1 multiplexer can be used to combine these four control signals into a single control bus. The select lines of the multiplexer can be used to select the appropriate control signal based on the instruction being executed.

In conclusion, multiplexing control signals in an ALU is an efficient way to manage the various operations performed by the ALU. It simplifies the control logic, reduces the number of control lines, and improves overall performance.

Would you like to delve deeper into a specific aspect of multiplexing or ALU control logic?

SUBJECT :- CA SUBJECT CODE-C2310C2

(5.) Explain how complement number system is useful in computer system. Discuss any one

complement number system with example.

Answers

Complement Number System: A Necessity in Computers

Complement number systems are essential in computer arithmetic for several

reasons: Efficient subtraction: Converting subtraction into addition simplifies

hardware design.

Negative number representation: They provide a straightforward method for representing negative numbers.

Overflow detection: Certain complement systems offer built-in overflow detection mechanisms.

1's Complement

Let's explore the 1's complement system as an example.

In a 1's complement system, a positive number is represented in its binary form as usual. To represent a negative number, we invert all bits of its positive counterpart.

Example:

Consider a 4-bit system.

Positive 5: 0101

Negative 5 (1's complement): 1010

Calculation with 1's complement:

Addition: Add the two numbers as binary numbers. If there's a carry-out from the most significant bit, add it back to the result (end-around carry).

Subtraction: Convert the subtrahend to its 1's complement and add it to the minuend. If there's no carry-out from the most significant bit, the result is negative; take the 1's complement of the result.

Example:

7 - 3 = 4

7 in binary: 0111

3 in binary: 0011

1's complement of 3: 1100

Addition: 0111 + 1100 = 10011 (end-around carry: 1)

Result: 0010 (which is 4 in binary)

Limitations of 1's complement:

There are two representations for zero (0000 and 1111), which can lead to complications.

The end-around carry operation adds complexity to the hardware.

Due to these limitations, the 2's complement system is more commonly used in modern computers.

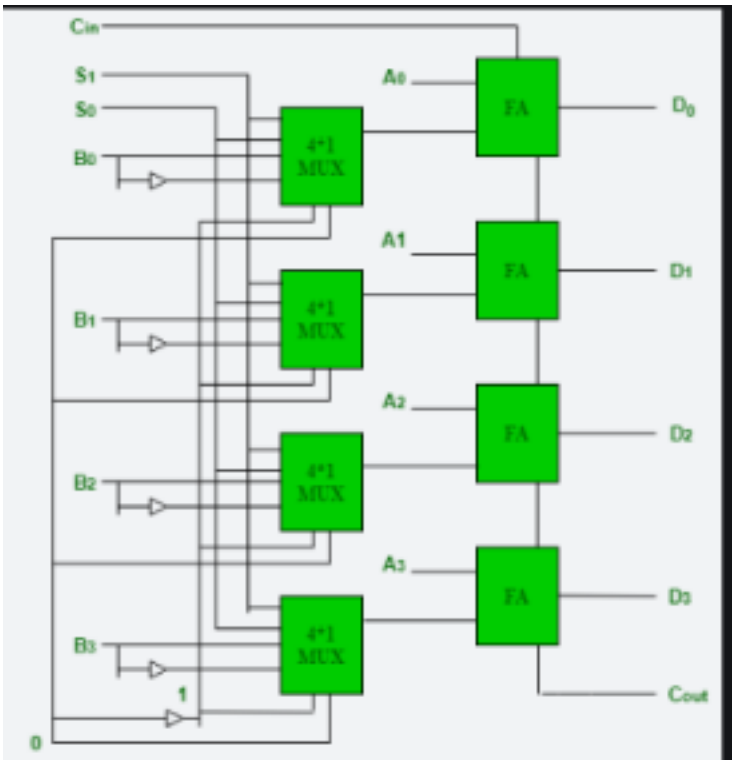
(6.)Draw the block diagram of 4-bit arithmetic circuit and explain it in detail.

Answers

4-Bit Arithmetic Circuit

A 4-bit arithmetic circuit is a digital circuit capable of performing basic arithmetic operations on two 4-bit binary numbers. It typically employs a combination of full adders and multiplexers to achieve this functionality.

Block Diagram



Components:

Inputs:

A: 4-bit input operand

B: 4-bit input operand

Cin: Carry-in bit

Select lines: Control the operation to be performed

Outputs:

Sum: 4-bit output result

Cout: Carry-out bit

Internal Components:

Four full adders: Perform binary addition on corresponding bits of A and B, along with the carry-in.

Multiplexers: Select the appropriate inputs for the full adders based on the control

signals. Explanation

The 4-bit arithmetic circuit operates as follows:

Input: The two 4-bit input operands, A and B, are applied to the circuit along with the carry in bit, Cin. The select lines determine the operation to be performed.

Multiplexer Selection: The multiplexers select the appropriate inputs for the full adders based on the control signals. This allows the circuit to perform different arithmetic operations.

Addition: The full adders perform binary addition on corresponding bits of A and B, along with the carry-in from the previous stage. The sum and carry-out of each full adder are passed to the next stage.

Output: The final sum is obtained from the outputs of the four full adders, and the carry-out from the last full adder is the overall carry-out of the circuit.

Operations

By appropriately setting the control signals, the 4-bit arithmetic circuit can perform various arithmetic operations, including:

Addition: A + B

Subtraction: A - B (using 2's complement)

Increment: A + 1

Decrement: A - 1

Other operations: Depending on the circuit's design, it might support additional operations like logical AND, OR, XOR, etc.

Additional Considerations

Carry-in and Carry-out: The carry-in and carry-out bits are essential for cascading multiple arithmetic circuits to handle larger numbers.

Overflow: The circuit should be designed to detect overflow conditions, indicating that the result is too large to be represented by the 4-bit output.

Speed and Area: The choice of full adder design and multiplexer implementation affects the circuit's speed and area.

By understanding the basic structure and operation of a 4-bit arithmetic circuit, you can build upon this foundation to create more complex arithmetic logic units (ALUs) for various digital

(7.)Explain shift micro operations and Draw neat and clean diagram for 4-bit combinational

circuit shifter.

SUBJECT :- CA SUBJECT CODE-C2310C2

Answers-

Shift Micro-Operations

Shift Micro-Operations are fundamental operations performed in digital systems where the bits in a register or accumulator are shifted to the left or right. These operations are essential for various tasks including arithmetic operations, data manipulation, and bit-level processing. Here's a detailed explanation of shift micro-operations:

1. **Logical Shift Left (LSL)**:

- **Operation**: Each bit in the register is shifted one position to the left. The vacated rightmost position is filled with zero.

- **Effect**: This operation multiplies the number by 2 for each shift position. It's useful in binary multiplication and for shifting data to prepare for further processing.

2. **Logical Shift Right (LSR)**:

- **Operation**: Each bit in the register is shifted one position to the right. The vacated leftmost position is filled with zero.

- **Effect**: This operation divides the number by 2 for each shift position, useful in binary division and for preparing data for subsequent operations.

3. **Arithmetic Shift Left (ASL)**:

- **Operation**: Similar to the logical shift left, but it also maintains the sign bit (for signed integers). Zero is filled in the vacated positions on the right.

- **Effect**: Multiplies the signed number by 2 while preserving its sign.

4. **Arithmetic Shift Right (ASR)**:

- **Operation**: Each bit in the register is shifted one position to the right. The vacated leftmost position is filled with the sign bit (the most significant bit).

- **Effect**: Divides the signed number by 2 while preserving the sign of the number.

5. **Circular Shift Left (Rotate Left)**:

- **Operation**: Bits are shifted to the left, and the leftmost bits are wrapped around and inserted into the rightmost positions.

- **Effect**: Useful in rotating data in a circular manner, maintaining the bit sequence but

changing the position.

6. **Circular Shift Right (Rotate Right)**:

- **Operation**: Bits are shifted to the right, and the rightmost bits are wrapped around and inserted into the leftmost positions.

- **Effect**: Similar to rotate left, this maintains the bit sequence while changing positions.

4-Bit Combinational Circuit Shifter

A 4-bit combinational circuit shifter can perform various shift operations like logical shifts, arithmetic shifts, and rotates. Below is a description and a diagram of a 4-bit combinational circuit shifter.

Components of the 4-Bit Shifter:

1. **Inputs**: 4-bit data input (D0, D1, D2, D3), Shift Control Signals, and Shift Amount (if variable shifting is used).

2. **Shift Control Signals**: Determine the type of shift operation (e.g., left shift, right shift, rotate).

3. **Logic Gates**: Implement the shifting logic.

4. **Multiplexers**: Select the appropriate data lines based on the shift

operation. #### Diagram

Here is a simple diagram of a 4-bit combinational shifter:

```plaintext
 +---+---+---+---+
Data | D0| D1| D2| D3|
 +---+---+---+---+
 | | | |
 V V V V
 +---+---+---+---+
Control| S0| S1| S2| S3|
```

```
+---+---+---+---+
| | | |
V V V V
+---+---+---+---+
Output | O0| O1| O2| O3|
+---+---+---+---+
```

```

#### Explanation of the Diagram:

1. **Data Inputs (D0, D1, D2, D3)**: Represent the 4-bit data that is to be shifted.
2. **Control Signals (S0, S1)**: Determine the type of shift operation. They can be used to select between:
   - **Logical Shift Left (LSL)**
   - **Logical Shift Right (LSR)**
   - **Arithmetic Shift Right (ASR)**
   - **Rotate Left (ROL)**
   - **Rotate Right (ROR)**
3. **Logic Implementation**: The shifter uses combinational logic circuits like multiplexers and logic gates to perform the required shift operation based on the control signals.

#### Functionality of Shifter:

- **For Logical Shifts**: Use gates to shift bits left or right, and insert zeroes into the vacant positions.
- **For Arithmetic Shifts**: Similar to logical shifts, but ensure the sign bit is handled correctly for right shifts.
- **For Rotations**: Implement the wrapping-around effect by using additional logic to move bits from one end to the other.

(8. )Explain hardware implementation of common bus system using three state buffers. Mention assumptions if required.

Answers
```

A **common bus system** is a fundamental design used in digital systems to enable multiple devices or components to share a single communication path, or bus. The hardware implementation of a common bus system using three-state buffers is a common approach to

SUBJECT :- CA SUBJECT CODE-C2310C2 manage data transfer between multiple devices and to avoid

conflicts on the bus.

Hardware Implementation of a Common Bus System

Overview

In a common bus system, multiple devices (such as CPUs, memory units, and I/O peripherals) are connected to a shared bus, which consists of data lines, address lines, and control lines. The three-state buffer plays a crucial role in this implementation by managing the data flow and ensuring that only one device communicates with the bus at any given time.

Key Components

1. **Data Bus**: A set of lines used for transferring data between devices. The width of the data bus (e.g., 8-bit, 16-bit, 32-bit) depends on the system design.

2. **Address Bus**: Lines used to specify the address of the device or memory location involved in the data transfer.

3. **Control Bus**: Lines used to send control signals that manage read/write operations and device selection.

4. **Three-State Buffers**: These buffers are used to interface devices with the bus. Each device has its own three-state buffer connected to the data bus.

Operation of Three-State Buffers

1. **Three States**:

- **High (Logic 1)**: The buffer drives the bus to a high voltage level.

- **Low (Logic 0)**: The buffer drives the bus to a low voltage level.

- **High Impedance (Z)**: The buffer disconnects itself from the bus, effectively leaving the bus in a floating state. This allows other devices to drive the bus without conflicts.

2. **Control Signals**: Each buffer has control inputs that determine its state. These inputs

typically include:

- **Enable Signal**: Determines whether the buffer should drive the bus or be in a high impedance state.

- **Data Input**: The data that will be placed on the bus if the buffer is enabled.

Implementation Steps

1. **Connecting Buffers**:

- Each device has a three-state buffer connected to the data bus. The buffer can be enabled or disabled based on the control signals.

- The address bus and control bus lines are used to select which device's buffer should be enabled at any given time.

2. **Control Signals**:

- **Address Decoding**: Use address decoding logic to determine which device is being accessed. This logic generates enable signals for the appropriate device based on the address lines.

- **Read/Write Control**: Signals that control whether data is being read from or written to the bus.

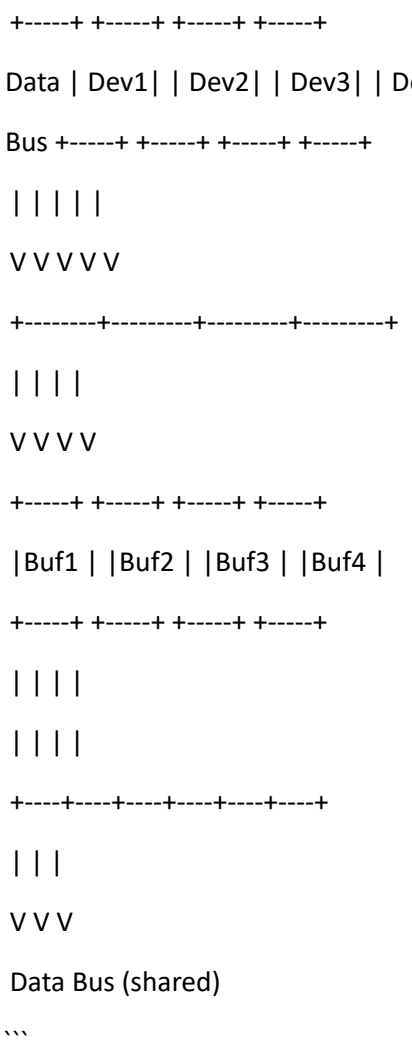
3. **Data Transfer**:

- **Write Operation**: When a device writes data to the bus, its buffer is enabled, and the data is placed on the bus. Other buffers are in a high-impedance state.

- **Read Operation**: When a device reads data from the bus, it will enable its buffer to receive data from the bus.

Diagram of a 4-Device Common Bus System Using Three-State Buffers

Here's a simplified diagram showing a 4-device common bus system with three-state buffers:



Assumptions

1. **Addressing**: A proper address decoding scheme is in place to select the active device and enable the corresponding buffer.

2. **Control Signals**: Appropriate read/write control signals are used to manage data direction and ensure correct data flow.

3. **Timing**: The system's timing must be managed to ensure that data is stable on the bus during read/write operations and that devices are properly synchronized.

Summary

The implementation of a common bus system using three-state buffers involves connecting multiple devices to a shared bus, with each device having a three-state buffer to control its access to the bus. The buffers are controlled by enable signals, which ensure that only one

device can drive the bus at any given time, preventing conflicts and ensuring proper data transfer.

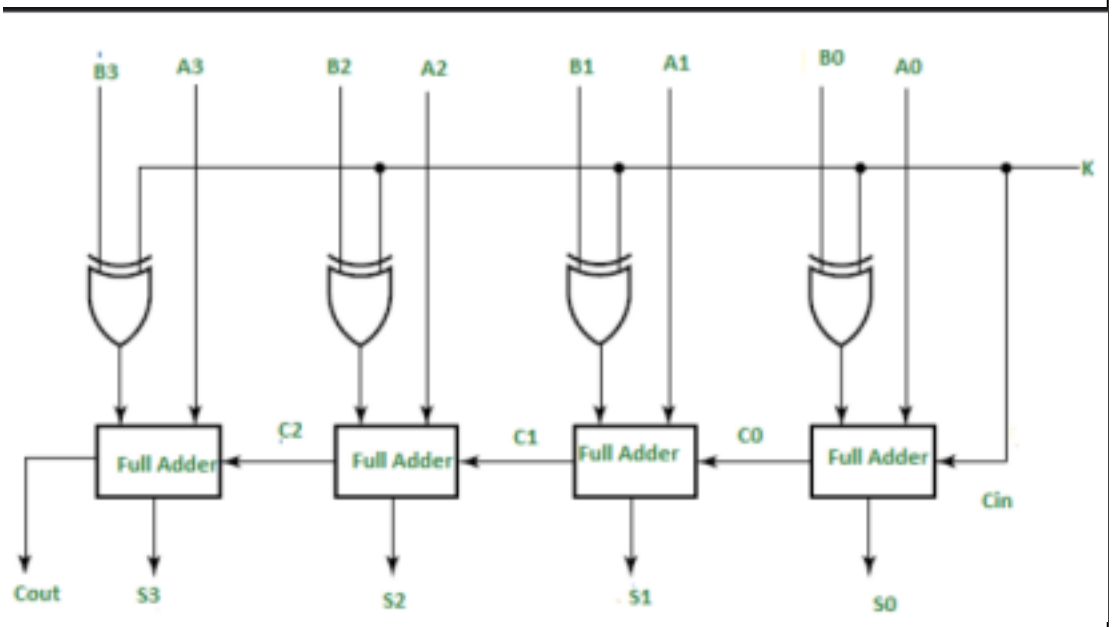
(9.)Explain 4-bit adder-subtractor with diagram.

Answers

4-Bit Adder-Subtractor

A 4-bit adder-subtractor is a digital circuit capable of performing both addition and subtraction operations on two 4-bit binary numbers. It's a versatile component often found in arithmetic logic units (ALUs).

Circuit Diagram



Explanation

The circuit primarily consists of:

Four full adders: These perform the basic addition operation.

XOR gates: Used to complement the subtrahend when performing

subtraction. Control line (K): Determines whether addition or subtraction is

performed. Operation:

Addition:

When the control line (K) is 0, the XOR gates output the original B

input. The full adders perform normal addition: Sum = A + B + Cin.

Subtraction:

When the control line (K) is 1, the XOR gates output the complement of B (B'). The full

adders perform A + B' + 1, which is equivalent to A - B. The initial '1' is provided as

the carry-in (Cin) to the first full adder.

Detailed Explanation

XOR gates: These gates are used to implement the subtraction operation. When K=0, the output of the XOR gate is B, and when K=1, the output is B'.

Full adders: There are four full adders, each taking three inputs: A, B (or B'), and Cin. The output of each full adder is the sum bit and the carry-out bit. The carry-out of the previous full adder becomes the carry-in for the next.

Control line (K): This single bit determines the operation. When K=0, the circuit performs addition, and when K=1, it performs subtraction.

By controlling the value of K, we can effectively use the same hardware for both addition and subtraction, making it an efficient design.

(10.)Explain floating point representation.

Answers

Floating-Point Representation

Floating-point representation is a method for representing real numbers in a computer system. It's analogous to scientific notation, where numbers are expressed as a significand (or mantissa) multiplied by a base raised to an exponent.

Components of Floating-Point Representation

A floating-point number typically consists of three parts:

Sign bit: Indicates whether the number is positive or negative.

Exponent: Represents the power of the base (usually 2 in computers).

Mantissa (or significand): The digits of the number.

Example:

Consider the decimal number 123.45. In scientific notation, it can be expressed as 1.2345 * 10^2. In a floating-point representation, this would be similar, with a base of 2 instead of 10.

IEEE 754 Standard

To ensure compatibility and accuracy, the IEEE 754 standard is widely used for floating-point representation. It defines different formats for representing floating-point numbers, such as single-precision (32 bits) and double-precision (64 bits).

Key points of IEEE 754:

Normalized form: The mantissa is always in the form 1.xxxxx (where x's are fractional bits), except for special cases like zero or denormalized numbers.

Bias: The exponent is stored as a biased value to allow for negative exponents.

Special values: The standard defines representations for special values like infinity, negative infinity, and NaN (Not a Number).

SUBJECT :- CA SUBJECT CODE-C2310C2

Advantages of Floating-Point Representation

Wide range of values: Can represent very small and very large numbers.

Efficient storage: Compared to fixed-point representation, floating-point often requires less storage for the same range of values.

Limitations of Floating-Point Representation

Precision loss: Due to the finite number of bits used for the mantissa, there can be a loss of precision in calculations.

Rounding errors: Arithmetic operations on floating-point numbers can introduce rounding errors.

Special cases: Handling special values like infinity and NaN can be complex.

(11.)What is a Digital Computer System? Explain the role of binary number system in it.

Answers

Digital Computer System

A digital computer system is an electronic device capable of performing arithmetic and logical operations automatically on data. It operates on the principle of digital signals, which represent information using discrete values, typically 0s and 1s.

Key components of a digital computer system:

Input devices: Used to feed data into the computer (keyboard, mouse, scanner, etc.)

Output devices: Display or produce the processed data (monitor, printer, speakers,

etc.)

Central Processing Unit (CPU): The brain of the computer, responsible for executing instructions.

Memory: Stores data and instructions for the CPU.

Storage devices: Long-term storage for data (hard drives, SSDs, etc.)

Role of Binary Number System

The binary number system is the foundation of digital computers. It uses only two digits, 0 and 1, to represent information. This simplicity makes it ideal for implementation in electronic circuits, where two voltage levels (high and low) can easily represent these digits.

Key roles of binary system in computers:

Data representation: All data, including text, numbers, images, and audio, is converted into binary format for processing and storage.

Arithmetic operations: The CPU performs arithmetic operations (addition, subtraction, multiplication, division) using binary logic.

Control signals: Binary signals control the operation of different computer components.

Memory addressing: Memory locations are identified using binary addresses.

In essence, the binary number system provides a universal language that computers can understand and process efficiently. It's the bridge between the human world and the electronic world of computers.

(12.)Design a digital circuit for 4-bit binary adder.

Answers-

4-Bit Binary Adder

Understanding the Basics

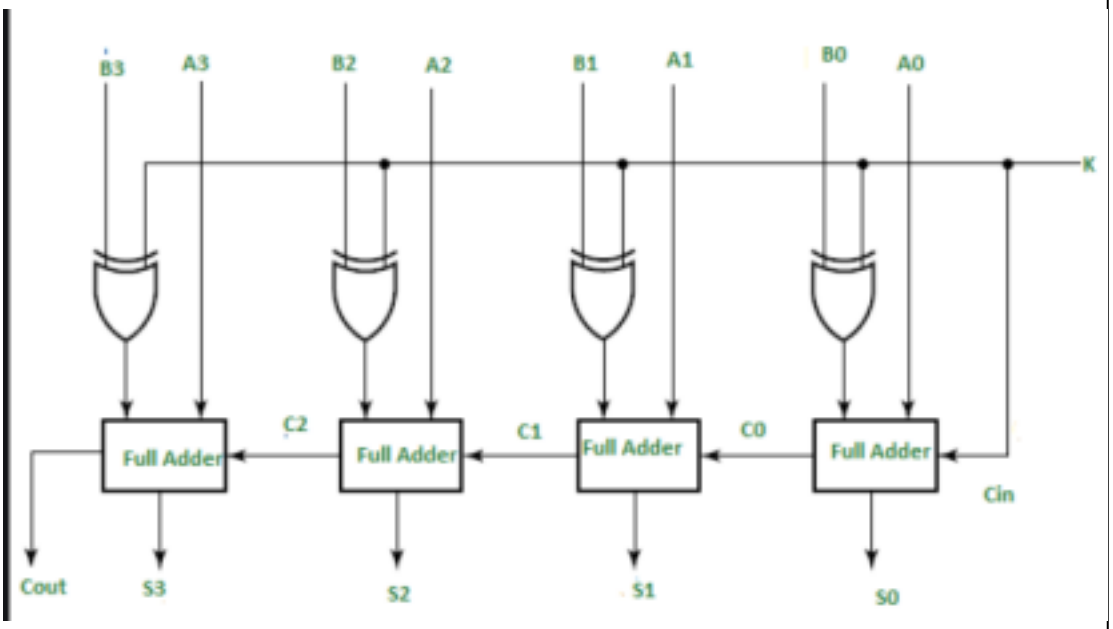
Before designing the circuit, let's understand the building blocks:

* **Half-Adder:** Adds two single-bit numbers (A and B) and produces a sum (S) and a carry-out (Cout).

* **Full-Adder:** Adds three single-bit numbers (A, B, and Cin) and produces a sum (S) and a carry-out (Cout).

A 4-bit adder consists of four full adders connected in series. The carry-out of one full adder becomes the carry-in for the next.

Circuit Design



Components:

* Four full adders

* Input lines: A3, A2, A1, A0 (for the first operand)

* Input lines: B3, B2, B1, B0 (for the second operand)

* Output lines: S3, S2, S1, S0 (for the sum)

* Output line: Cout (for the carry-out)

Connections:

* The least significant bits (A0 and B0) are connected to the inputs of the first full adder.

* The carry-out of the first full adder is connected to the carry-in of the second full adder, and so on.

* The sum outputs of each full adder form the output bits S3, S2, S1, and

S0. * The carry-out of the fourth full adder is the final carry-out (Cout).

Explanation

1. The four full adders are connected in a cascaded manner.

2. The least significant bits (A0 and B0) are added by the first full adder, producing a sum bit (S0) and a carry-out (C0).

3. The carry-out (C0) from the first full adder is used as the carry-in for the second full adder, along with the corresponding bits A1 and B1. This process continues for the remaining full adders.

4. The final sum is obtained from the sum outputs of the four full adders (S3, S2, S1,

S0). 5. The carry-out from the last full adder is the overall carry-out of the 4-bit

adder.

Truth Table

A truth table can be created to verify the correct operation of the 4-bit adder. It would list all possible combinations of the input bits (A3, A2, A1, A0, B3, B2, B1, B0) and the corresponding output bits (S3, S2, S1, S0, Cout).

By following this design, you can create a 4-bit binary adder that accurately performs addition operations on two 4-bit numbers.

(13.)Represent (8620)10 in (1) binary (2) Excess-3 code and (3) 2421 code.

Answers

Representing (8620)10 in Different Number Systems

SUBJECT :- CA SUBJECT CODE-C2310C2

(1) Binary Representation

To convert decimal to binary, we repeatedly divide the decimal number by 2 and record the remainders. The remainders, read from bottom to top, form the binary equivalent.

8620 / 2 = 4310, remainder 0

4310 / 2 = 2155, remainder 0

2155 / 2 = 1077, remainder 1

1077 / 2 = 538, remainder 1

538 / 2 = 269, remainder 0

269 / 2 = 134, remainder 1

134 / 2 = 67, remainder 0

67 / 2 = 33, remainder 1

33 / 2 = 16, remainder 1

16 / 2 = 8, remainder 0

8 / 2 = 4, remainder 0

4 / 2 = 2, remainder 0

2 / 2 = 1, remainder 0

1 / 2 = 0, remainder 1

So, (8620)10 in binary is (10000110010100)2.

(2) Excess-3 Code

In Excess-3 code, each decimal digit is represented by its binary equivalent plus

3. 8 in Excess-3: 8 + 3 = 11, binary of 11 is 1011

6 in Excess-3: 6 + 3 = 9, binary of 9 is 1001

2 in Excess-3: 2 + 3 = 5, binary of 5 is 0101

0 in Excess-3: 0 + 3 = 3, binary of 3 is 0011

So, (8620)10 in Excess-3 code is (1011100101010011)2.

(3) 2421 Code

In 2421 code, each decimal digit is represented by a 4-bit code based on the weights 2, 4, 2, and 1.

8 in 2421: 1000

6 in 2421: 1100

2 in 2421: 0010

0 in 2421: 0000

So, (8620)10 in 2421 code is (1000110000100000)2.

Note: All the results are in binary format.

(14.)Explain selective set, selective complement and selective clear.

Answers

Selective Set, Selective Complement, and Selective Clear

These are bit-wise operations performed on registers in a computer system. They allow for precise manipulation of individual bits within a register based on a mask.

Selective Set

Definition: Sets the bits in a register to 1 where there are corresponding 1's in a mask register.

Operation: A ← A OR B

Example:

A = 1010

B (mask) = 1100

Result: A = 1110

Selective Complement

Definition: Complements the bits in a register where there are corresponding 1's in a mask register.

Operation: A ← A XOR B

Example:

A = 1010

B (mask) = 1100

Result: A = 0110

Selective Clear

Definition: Clears (sets to 0) the bits in a register where there are corresponding 1's in a mask register.

Operation: A ← A AND (NOT B)

Example:

A = 1010

B (mask) = 1100

Result: A = 0010

Key points:

These operations are performed bit-wise, meaning each bit in the register is affected independently.

The mask register determines which bits are affected.

These operations are fundamental to many digital circuit designs and are used in various data manipulation tasks.

Applications:

Bit manipulation: Setting, clearing, or inverting specific bits in a register.

Data masking: Extracting or isolating specific parts of data.

Error correction: Correcting errors in data by selectively modifying bits.

By understanding these operations, you can effectively manipulate data at the bit level and implement various digital circuit functions.

(15.)How negative integer number represented in memory? Explain with suitable example.

Answers

Representing Negative Integers in Memory: Two's Complement

Two's complement is the most commonly used method for representing negative integers in computer systems. This method offers several advantages, including simplicity in arithmetic operations.

How it works:

Determine the bit width: The number of bits used to represent the integer. For example, a 4- bit system can represent numbers from -8 to 7.

Convert the positive number to binary: Convert the absolute value of the negative number to its binary equivalent.

Invert all bits: Change all 0s to 1s and 1s to 0s.

Add 1: Add 1 to the result from step 3.

Example:

Let's represent -5 in a 4-bit system.

Bit width: 4 bits

Binary equivalent of 5: 0101

Invert all bits: 1010

Add 1: 1010 + 1 = 1011

So, -5 is represented as 1011 in 2's complement form in a 4-bit system.

Why Two's Complement?

Simplicity of arithmetic operations: Addition, subtraction, multiplication, and division can be performed using the same hardware for both positive and negative numbers.

Unique representation of zero: There is only one representation for zero (0000).

Efficient hardware implementation: The logic for implementing two's complement arithmetic is relatively simple.

Key points:

The most significant bit (MSB) acts as a sign bit. A 0 indicates a positive number, and a 1 indicates a negative number.

The range of representable numbers for an n-bit system is from -2^(n-1) to 2^(n-1) - 1.

By understanding two's complement representation, you can effectively work with negative numbers in computer systems and comprehend the underlying principles of computer arithmetic.

(16.)Explain Micro operation.

Answers

Micro-operations

Micro-operations are the fundamental, low-level operations performed by a processor. They are the building blocks of more complex instructions. These operations are typically simple and involve manipulating data within registers or between registers and memory.

Types of Micro-operations:

Data Transfer:

Moving data from one register to another.

Loading data from memory into a register.

Storing data from a register into memory.

Arithmetic Operations:

Addition, subtraction, multiplication, and division.

Incrementing or decrementing the value of a register.

Logical Operations:

AND, OR, NOT, XOR, and other logical functions.

Comparing data for equality, greater than, less than, etc.

Shift Operations:

Shifting bits within a register (left shift, right shift, circular shift).

Control Operations:

Modifying the program counter.

SUBJECT :- CA SUBJECT CODE-C2310C2

Activating or deactivating control signals.

Example:

Consider a simple instruction like "ADD R1, R2". This instruction can be broken down into several micro-operations:

Fetch the instruction from memory.

Decode the instruction to determine the operation and operands.

Read the contents of registers R1 and R2.

Perform the addition operation using the ALU.

Store the result in register R1.

Each of these steps involves one or more micro-operations.

Importance of Micro-operations:

Understanding processor architecture: Micro-operations provide insights into the internal workings of a CPU.

Designing digital systems: They are essential for designing control units and data paths in digital circuits.

Optimizing performance: Analyzing micro-operations can help identify performance bottlenecks and optimization opportunities.

In essence, micro-operations are the foundation upon which higher-level instructions are built. By understanding these basic operations, one can gain a deeper appreciation for the complexity and efficiency of modern computer systems.

(17.)What does this mean: R2 ← R1?

Answers

R2 ← R1

This notation means that the contents of register R1 are transferred to register

R2. R1 and R2 are names of registers within a computer's processing unit.

← is the transfer operator, indicating the direction of data movement.

In simpler terms, whatever value is stored in register R1 will be copied and placed into register R2. The original content of R1 remains unchanged.

This is a basic example of a register transfer operation, which is a fundamental operation in computer architecture.

(18.)What does this mean: T0: R4 ← R0?

Answers

T0: R4 ← R0

Breakdown:

T0: This likely refers to a time step or cycle within a processor's operation. It indicates that the following action occurs at this specific point in time.

R4 ← R0: This part of the statement means that the contents of register R0 are transferred to register R4. The arrow indicates the direction of data movement.

Overall Meaning:

The statement "T0: R4 ← R0" signifies that at time step T0, the value stored in register R0 is copied and placed into register R4.

Example:

If register R0 contains the value 10, after executing this statement at time T0, register R4 will also contain the value 10.

This type of instruction is a fundamental operation in computer architecture, often used to manipulate data within the processor.

(19.)What is a Bus?

Answers

Bus in Computer Architecture

A bus is a communication system that transfers data between components inside a computer, or between computers. Think of it as a highway that carries information between different parts of a computer.

Key Components of a Bus:

Data lines: Carry the actual data being transferred.

Address lines: Specify the location (memory address) where data is to be sent or retrieved.

Control lines: Coordinate the data transfer process, including start, stop, and acknowledge signals.

Types of Buses:

System bus: Connects the CPU, memory, and input/output devices.

Address bus: Carries memory addresses from the CPU to the memory.

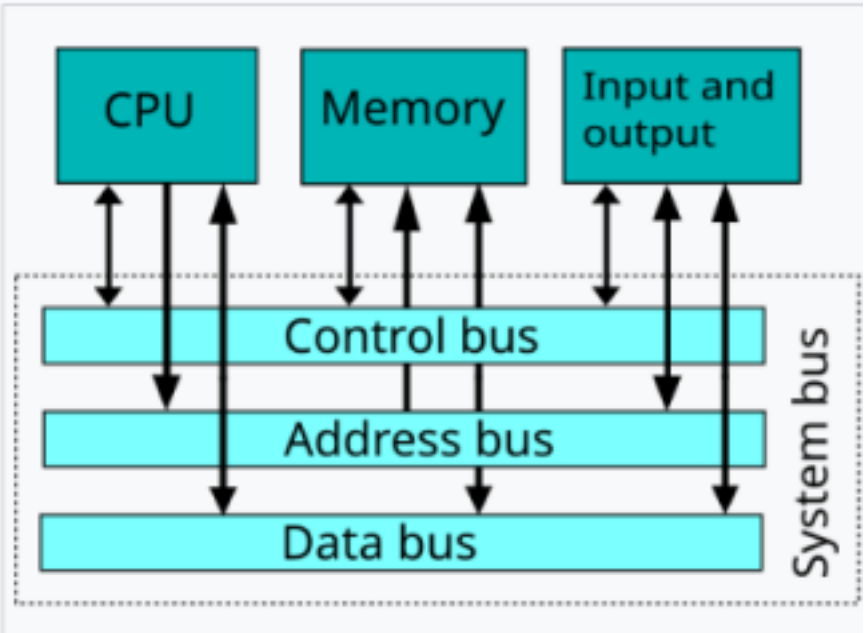
Data bus: Transfers data between components.

Control bus: Synchronizes the actions of different components.

Peripheral bus: Connects peripheral devices (like printers, keyboards) to the computer. Importance of Buses:

Efficient data transfer: Buses enable high-speed data movement within a computer system. Shared resource: Components can share the bus to communicate with each other. Scalability: Buses allow for expansion of the system by adding new components.

SUBJECT :- CA SUBJECT CODE-C2310C2



In essence, a bus is the backbone of a computer system, facilitating communication and data transfer between its various components.

(20.)What is an ALU?

Answers

ALU stands for Arithmetic Logic Unit. It's the computational brain of a computer's central processing unit (CPU).

What does it do?

Performs arithmetic operations: addition, subtraction, multiplication,

division. Executes logical operations: AND, OR, NOT, XOR.

Handles bitwise operations: shifting, rotating, comparing bits.

Essentially, any mathematical or logical calculation a computer performs happens within the ALU. It's a crucial component that enables the computer to process information and carry out tasks.

(21.)Represent the following conditional control statement(s) by two register transfer statements with control function. If (P=1) then (R1 ← R2) else if (Q=1) then (R1 ← R3).

Answers

Representing the Conditional Control Statement

Understanding the Problem

We're tasked with converting the following conditional statement into register transfer statements with control functions:

If (P=1) then (R1 ← R2) else if (Q=1) then (R1 ← R3)

Register Transfer Statements with Control Functions

To represent this conditional logic, we can use two register transfer statements with control functions:

Statement 1:

T1: R1 ← R2 when P = 1

This statement means: If P equals 1, then transfer the contents of register R2 to register R1 at time T1.

Statement 2:

T2: R1 ← R3 when P'Q = 1

This statement means: If P is not equal to 1 (denoted by P') and Q equals 1, then transfer the contents of register R3 to register R1 at time T2.

Explanation

The control functions P = 1 and P'Q = 1 determine when the respective register transfers occur.

The use of P' (NOT P) ensures that the second condition is only evaluated when the first condition is false.

By using these two register transfer statements, we've effectively represented the given conditional control statement in a form suitable for hardware implementation.

Note: The specific timing (T1 and T2) would depend on the overall control unit design and the desired execution sequence.

(22.)State true or false: In binary number system, B - A is equivalent to B + A' + 1.

Answers

True.

The statement B - A is equivalent to B + A' + 1 is indeed true in the binary number system.

This is the fundamental principle of two's complement representation for negative numbers. By adding the one's complement (A') of a number to the other number (B) and then adding 1, we effectively subtract A from B.

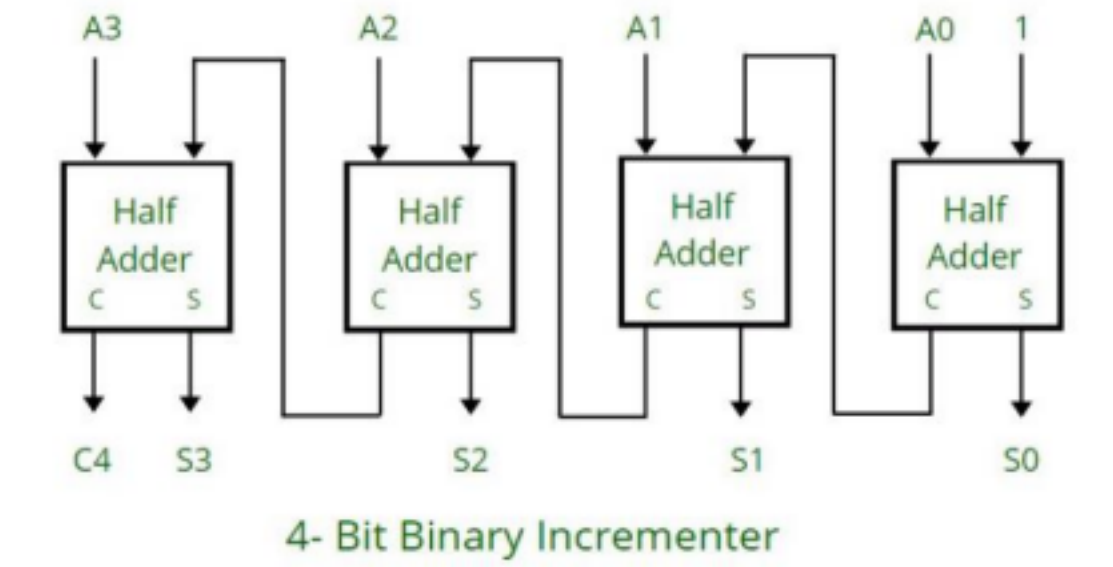
(23.)Draw a diagram of 4-bit binary incrementer and explain it briefly.

Answers

4-Bit Binary Incrementer

A 4-bit binary incrementer adds 1 to a 4-bit binary number. It's essentially a simple adder circuit with one input permanently set to 1.

Diagram:



4- Bit Binary Incrementer

Explanation:

The circuit consists of four half-adders connected in series.

The least significant bit (LSB) half-adder has one input permanently connected to logic 1.

The other input to each half-adder is connected to the corresponding bit of the input number.

The carry output of each half-adder is connected to the carry input of the next higher-order half-adder.

The sum outputs of the four half-adders form the incremented output.

Operation:

The least significant bit (LSB) half-adder adds 1 to the LSB of the input number, producing the sum bit and a carry.

The carry from the previous half-adder is added to the next bit of the input number in the next half-adder.

This process continues for all four bits.

The final sum outputs represent the incremented value.

In essence, a 4-bit binary incrementer efficiently adds 1 to a 4-bit binary number using a cascade of half-adders.