Q.1 In a digital computer system having 32 registers of size 64 bits each, a common bus system has to be designed using multiplexers. Determine the following for the common bus system design ?

1. How many multiplexers are required?

In a common bus system, each register needs to be connected to the bus. Assuming each register has a separate path to the common bus, we would need one multiplexer per register. Since there are 32 registers, you would need 32 multiplexers.

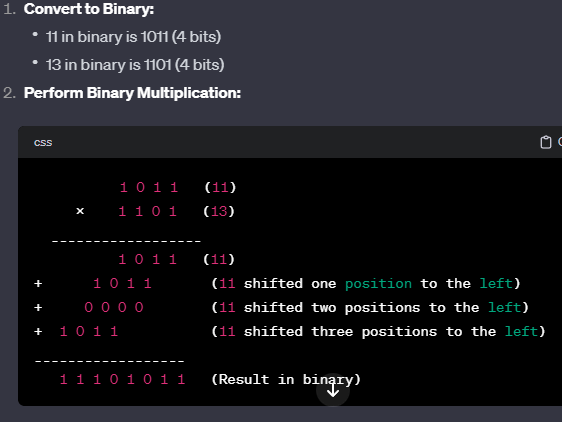
1. What is the size of each multiplexer?

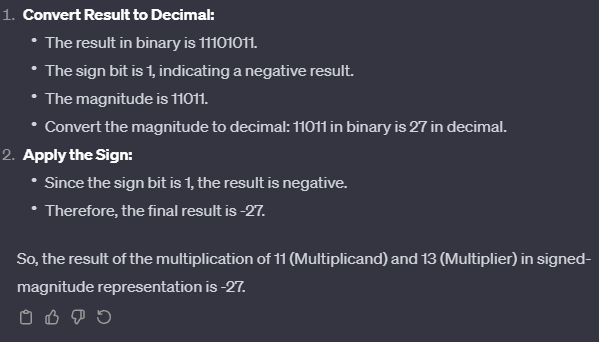
The size of each multiplexer is determined by the number of input lines it has. The number of input lines required for each multiplexer is equal to the number of bits in each register. Each multiplexer should be able to handle the data from one register. Since each register is 64 bits, each multiplexer should have 64 input lines.

1. How many selection inputs are there in each multiplexer?

The number of selection inputs (also known as control inputs) for a multiplexer is determined by the formula 2^n, where n is the number of input lines. In this case, each multiplexer has 64 input lines, so it would require 6 selection inputs (2^6 = 64). These selection inputs determine which of the 64 input lines is routed to the output.

Q.2 Apply binary multiplication method for signed- magnitude numbers and perform the multiplication of 11 as Multiplicand and 13 as Multiplier





Q.3 Give the hardware implementation of the following register transfer language (RTL) statement

using the block diagram

a) w + P: R5  R7, R7  R5

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

| | | |

---->| R7 | | R5 |<----

| | | | | |

| +--------+ +--------+ |

| |

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

| |

| |

V V

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

| |

| Addition |

| |

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

|

|

V

+--------+

| |

| R7 |

| |

+--------+

b) abc + a'bc +abc': AR  AR + AC

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

| | | | | |

-----> | AR | +---->| AC | +---->| NOT | +-----+

| | | | | | |

+--------+ +--------+ +--------+ |

| |

V V

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

| | | |

| Multiplex | | |

| | | AND |

+-----------------+ | |

| |

V V

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

| | | |

| OR | | OR |

| | | |

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

| |

V V

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

| | | |

| AR | | AR |

| | | |

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

Q.4 Perform conversion of the following arithmetic expressions from infix notation to reverse polish notation (RPN) for part (a), (b), and (c) and vice-versa in remaining parts of the question.

a) A + B\*[C\*D + E\*(F + G)]

b) A\*[B + C\*(D + E)] / (F\*(G + H))

c) P\*Q + R\*S + Y\*Z

d) A B C D E + \* - /

e) A B C D E \* / - +

f) A B C \* / D - E F / +

g) A B C D E F G + \* + \* + \*

a) https://www.quora.com/How-do-I-convert-the-following-infix-expression-A-B-C-D-E-F-G-in-reverse-Polish-notation

b) Infix to Reverse Polish Notation (RPN):

Add D and E.

Multiply C with the result of step 1.

Add B to the result of step 2.

Multiply A with the result of step 3.

Add G and H.

Multiply F with the result of step 5.

Divide the result of step 4 by the result of step 6.

RPN: D E + C \* B + A \* G H + F \* /

c) Infix to Reverse Polish Notation (RPN):

Multiply P and Q.

Multiply R and S.

Add the results of steps 1 and 2.

Multiply Y and Z.

Add the result of step 3 to the result of step 4.

RPN: P Q \* R S \* + Y Z \* +

d) Reverse Polish Notation (RPN) to Infix:

Push A.

Push B.

Push C.

Push D.

Push E.

Pop E and D, then perform E + D.

Push the result of step 6.

Pop the result of step 7 and C, then perform C \* (E + D).

Push the result of step 8.

Pop the result of step 9 and B, then perform B + C \* (E + D).

Push the result of step 10.

Pop the result of step 11 and A, then perform A / (B + C \* (E + D)).

Infix: A / (B + C \* (E + D))

e) Reverse Polish Notation (RPN) to Infix:

Push A.

Push B.

Push C.

Push D.

Push E.

Pop E and D, then perform E \* D.

Push the result of step 6.

Pop the result of step 7 and C, then perform C / (E \* D).

Push the result of step 8.

Pop the result of step 9 and B, then perform B - C / (E \* D).

Push the result of step 10.

Pop the result of step 11 and A, then perform A + (B - C / (E \* D)).

Infix: A + (B - C / (E \* D))

f) Reverse Polish Notation (RPN) to Infix:

Push A.

Push B.

Push C.

Pop C and B, then perform C / B.

Push the result of step 4.

Pop the result of step 5 and A, then perform A \* (C / B).

Push the result of step 6.

Pop D.

Pop E.

Pop F.

Perform E / F.

Perform D - (E / F).

Perform (A \* (C / B)) + (D - (E / F)).

Infix: (A \* (C / B)) + (D - (E / F))

g) Reverse Polish Notation (RPN) to Infix:

Push A.

Push B.

Push C.

Push D.

Push E.

Push F.

Push G.

Pop G and F, then perform G + F.

Push the result of step 8.

Pop the result of step 9 and E, then perform E \* (G + F).

Push the result of step 10.

Pop the result of step 11 and D, then perform D \* (E \* (G + F)).

Push the result of step 12.

Pop the result of step 13 and C, then perform C \* (D \* (E \* (G + F))).

Push the result of step 14.

Pop the result of step 15 and B, then perform B + C \* (D \* (E \* (G + F))).

Push the result of step 16.

Pop the result of step 17 and A, then perform A \* (B + C \* (D \* (E \* (G + F)))).

Infix: A \* (B + C \* (D \* (E \* (G + F))))

Q.5 An 8-bit register contains the binary value 10011100. What is the register value after arithmetic shift right? Starting from the initial number 10011100, determine the register value after an arithmetic shift left, and state whether there is an overflow.

Let's perform an arithmetic shift right on the 8-bit binary value 10011100.

Arithmetic Shift Right (ASR):

For an arithmetic shift right, the most significant bit (MSB) is shifted into the vacant bit positions, preserving the sign bit. In an 8-bit register, the MSB is the leftmost bit.

Starting value: 10011100

After one arithmetic shift right: 11001110

Arithmetic Shift Left (ASL):

For an arithmetic shift left, the least significant bit (LSB) is shifted into the vacant bit positions, and a 0 is shifted into the LSB.

Starting value: 10011100

After one arithmetic shift left: 00111000

Q.6 The four 8-bit registers along with their values are: AR = 11110010, BR = 11111111, CR = 10111001 and DR = 11101010. Determine the 8-bit values in each register after the execution of the following

sequence of microoperations.

a) AR  AR + BR

b) CR  CR ^ DR

c) BR  BR + 1

d) AR  AR – CR

a) AR ← AR + BR

AR=11110010+11111111AR=11110010+11111111

AR=11110010+11111111‾AR=11110010+11111111​

AR=101100101‾AR=101100101​

After the operation, AR becomes 1011001010110010.

b) CR ← CR ^ DR

CR=10111001 XOR 11101010CR=10111001XOR11101010

CR=01010011‾CR=01010011​

After the operation, CR becomes 0101001101010011.

c) BR ← BR + 1

BR=11111111+1BR=11111111+1

BR=100000000‾BR=100000000​

After the operation, BR becomes 0000000000000000 (since we are working with 8-bit registers).

d) AR ← AR - CR

AR=10110010−01010011AR=10110010−01010011

AR=10110010−01010011‾AR=10110010−01010011​

AR=01011111‾AR=01011111​

After the operation, AR becomes 0101111101011111.

So, after the sequence of microoperations:

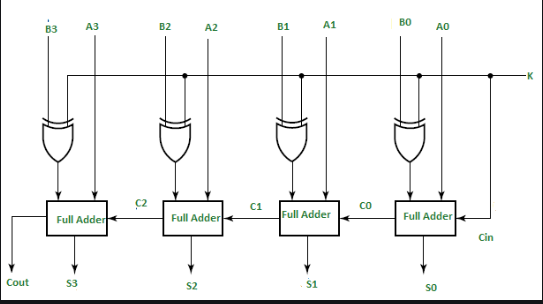
AR = 0101111101011111

BR = 0000000000000000

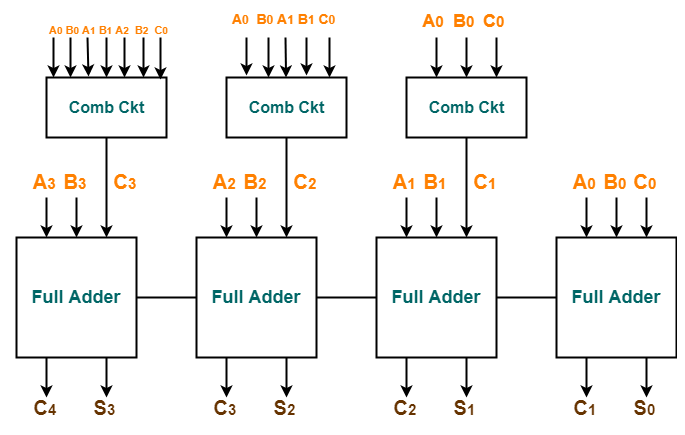
CR = 0101001101010011

DR = 1110101011101010 (unchanged)

Q.7 Design a 4-bit Adder/ Subtractor using parallel adder.



Q.8 Design 4 bit carry look ahead adder.



Q.9 Explain the microoperations involved for the PUSH and POP operations in a register stack and also for a memory stack with an example.

A stack is a data structure that follows the Last In First Out (LIFO) principle. This means that the last element added to the stack is the first element to be removed. Stacks are often used to implement function calls and return addresses.

There are two main operations that can be performed on a stack: push and pop. A push operation adds an element to the top of the stack, while a pop operation removes the element at the top of the stack.

The microoperations involved in a push operation on a register stack are as follows:

The stack pointer (SP) is incremented by .

The data to be pushed is stored in the register at the new SP address.

The microoperations involved in a pop operation on a register stack are as follows:

The data at the current SP address is stored in a register.

The SP is decremented by 1.

The microoperations involved in a push operation on a memory stack are as follows:

The stack pointer (SP) is incremented by 1.

The data to be pushed is stored in the memory location at the new SP address.

The microoperations involved in a pop operation on a memory stack are as follows:

The data at the current SP address is stored in a register.

The SP is decremented by 1.

The data in the register is returned.

Here is an example of how a stack can be used to implement a function call. When a function is called, the return address is pushed onto the stack. When the function returns, the return address is popped from the stack and the program execution continues at the point where the function was called.

// Push the return address onto the stack

push(return\_address);

// Call the function

function();

// Pop the return address from the stack

return\_address = pop();

Q.10 A computer is designed for 32-bit instructions and 12-bit addresses. If there are 250 two-address

instructions then calculate the maximum number of one-address instructions that can be generated?

In a computer system, the number of bits used for instructions and addresses determines the maximum number of unique instructions and addresses that can be represented. Let's calculate the maximum number of one-address instructions based on the given information:

Given:

Instruction size: 32 bits

Address size: 12 bits

Number of two-address instructions: 250

Two-Address Instructions:

A two-address instruction typically has two operands and an opcode. The size of each instruction is the sum of the sizes of the two addresses plus the opcode.

Therefore, the size of a two-address instruction is 2×Address size+Opcode size2×Address size+Opcode size.

Size of two-address instruction=2×12 bits+Opcode size=24 bits+Opcode size

Size of two-address instruction=2×12bits+Opcode size=24bits+Opcode size

Given that the size of a two-address instruction is 32 bits, we can find the size of the opcode:

24 bits+Opcode size=32 bits24bits+Opcode size=32bits

Opcode size=32 bits−24 bits=8 bitsOpcode size=32bits−24bits=8bits

Maximum Number of One-Address Instructions:

For one-address instructions, each instruction has one operand and an opcode. The size of each instruction is the sum of the size of the address and the opcode.

Therefore, the size of a one-address instruction is Address size+Opcode sizeAddress size+Opcode size.

Size of one-address instruction=12 bits+8 bits=20 bits

Size of one-address instruction=12bits+8bits=20bits

Now, we can calculate the maximum number of unique one-address instructions based on the instruction size:

Number of one-address instructions=Instruction size

Size of one-address instructionNumber of one-address instructions=Size of one-address instructionInstruction size​

Number of one-address instructions=32 bits20 bits

Number of one-address instructions=20bits32bits​

Number of one-address instructions≈1.6

Number of one-address instructions≈1.6

Since the number of instructions must be a whole number, we round down to the nearest integer:

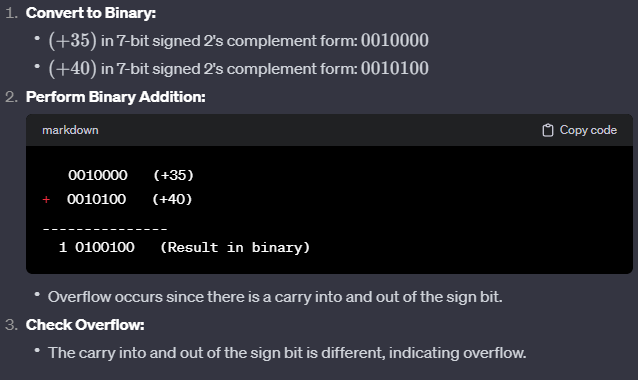
Maximum number of one-address instructions=1Maximum number of one-a

address instructions=1

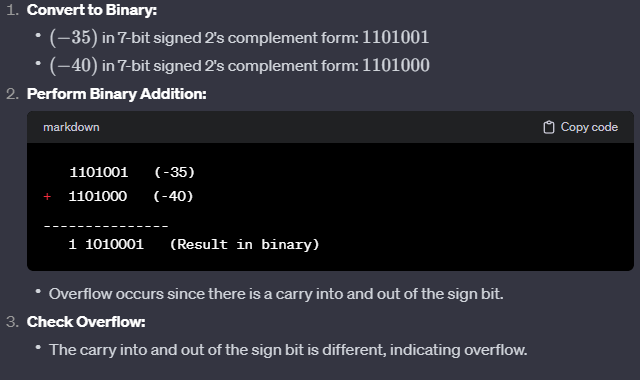
Therefore, the maximum number of one-address instructions that can be generated is 1.

Q.11 Execute the arithmetic calculations below using binary numbers and negative numbers represented in signed 2’s complement form. Employ seven bits to represent each number along with its sign. For each case, ascertain the presence of overflow by examining the carries into and out of the sign bit:

1. (+35) + (+40)



b) (-35) + (-40)



Q.12 Design and explain the working of 4-bit Arithmetic Circuit that can perform any of the following

micro-operations: addition, subtraction, increment, and decrement.

A 4-bit Arithmetic Circuit (AC) is a digital circuit that can perform the four basic arithmetic operations: addition, subtraction, increment, and decrement. It is composed of four full adders and four multiplexers. The full adders are used to perform the actual arithmetic operations, while the multiplexers are used to select which operation to perform.

The AC has four inputs, A, B, Cin, and Sel. The inputs A and B are the two operands, Cin is the carry-in, and Sel is the selection signal. The output of the AC is the sum or difference of A and B, depending on the value of Sel.

The following table shows the truth table for the AC:

Sel |Operation  
------- |--------  
00 |Add  
01 |Subtract  
10 |Increment  
11 |Decrement

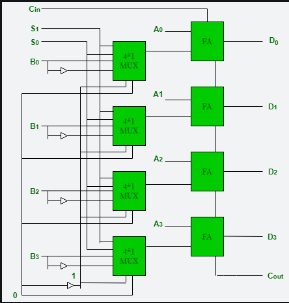
When Sel is 00, the AC performs addition. The full adders are connected in a ripple-carry fashion, so the carry-out of one full adder is connected as the carry-in to the next. The sum of A and B is produced at the output of the most significant full adder.

When Sel is 01, the AC performs subtraction. The full adders are connected in the same way as for addition, but the B input is inverted. This causes the AC to subtract B from A. The difference of A and B is produced at the output of the most significant full adder.

When Sel is 10, the AC performs increment. The full adders are connected in the same way as for addition, but the B input is set to zero. This causes the AC to add 1 to A. The incremented value of A is produced at the output of the most significant full adder.

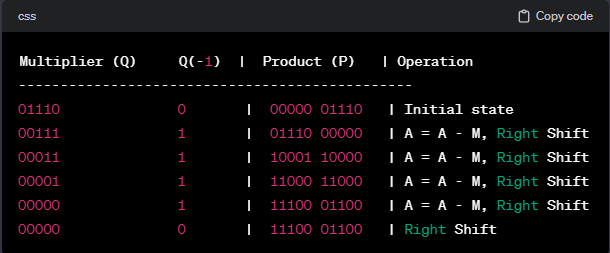
When Sel is 11, the AC performs decrement. The full adders are connected in the same way as for subtraction, but the B input is set to all ones. This causes the AC to subtract 1 from A. The decremented value of A is produced at the output of the most significant full adder.

The 4-bit AC is a versatile circuit that can be used to perform a variety of arithmetic operations. It is commonly used in microprocessors and other digital devices.

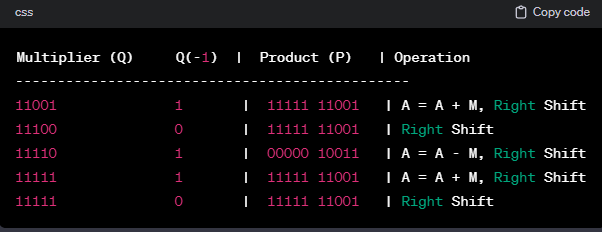


Q.13 Perform multiplication of 5-bit numbers using Booth's Algorithm of the following

1. (+14) X (-14)



1. (-7) X (+3)



c) (10010)2 X (10100)2



Q.14 Explain the various addressing modes with the help of a numerical example.

Addressing modes in computer architecture define how operands are specified in instructions. Different addressing modes provide flexibility in programming and allow for efficient use of resources. Here are some common addressing modes explained with numerical examples:

Immediate Addressing Mode:

In immediate addressing, the operand is directly specified in the instruction.

Example: MOV A, #5 (Move the value 5 to register A).

Register Addressing Mode:

The operand is in a processor register.

Example: ADD B, C (Add the content of register C to register B).

Direct Addressing Mode:

The operand's memory address is given directly in the instruction.

Example: LOAD A, 2000 (Load the content of memory address 2000 into register A).

Indirect Addressing Mode:

The operand's address is held in another register or memory location.

Example: LOAD A, (B) (Load the content of the memory address stored in register B into register A).

Register Indirect Addressing Mode:

The address is specified indirectly through a register.

Example: MOV A, (R1) (Move the content of the memory location whose address is stored in register R1 to register A).

Indexed Addressing Mode:

An index or offset is added to a base address to get the effective address.

Example: ADD A, B[3] (Add the content of memory location at address B + 3 to register A).

Base-Register Addressing Mode:

Similar to indexed addressing but with a fixed register as the base.

Example: LOAD A, 1000(B) (Load the content of memory address 1000 + content of register B into register A).

Relative Addressing Mode:

The operand is at a location relative to the program counter.

Example: JUMP 10 (Jump to the instruction 10 addresses away from the next instruction).

Stack Addressing Mode:

Operands are implicitly on the top of the stack.

Example: POP A (Pop the top of the stack and store it in register A).

Auto-increment and Auto-decrement Addressing Modes:

The address is incremented or decremented automatically after each access.

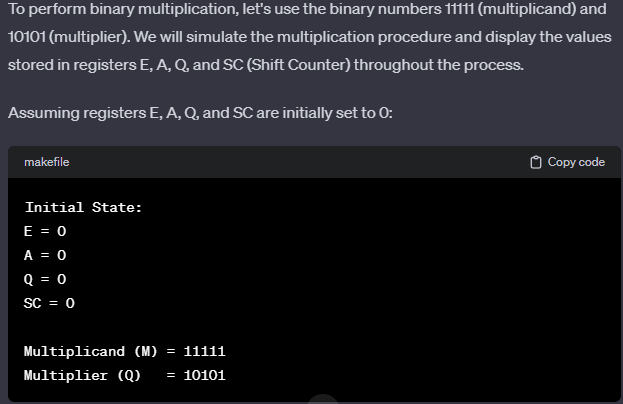
Example: LOAD A, (B)+ (Load the content of the memory location addressed by B into register A and increment the content of B).

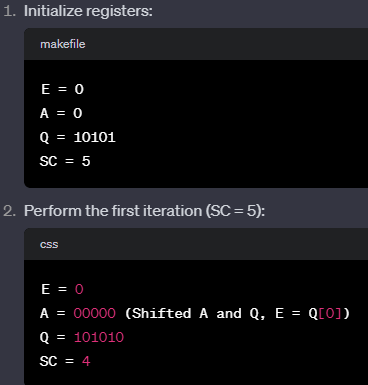
Scaled Addressing Mode:

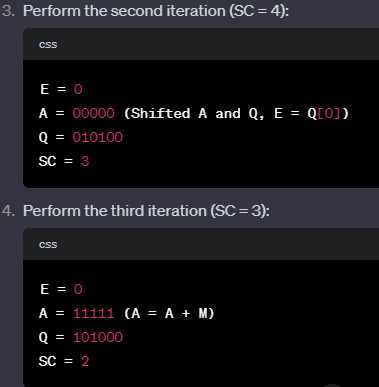
The index is multiplied by a scaling factor before being added to the base address.

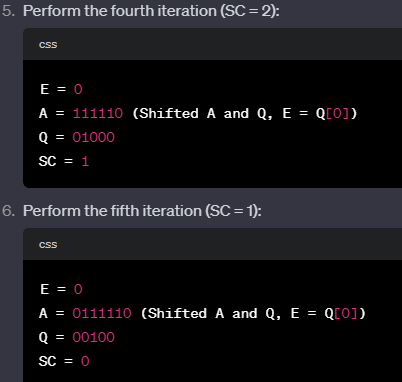
Example: MOV A, B[2 \* C] (Move the content of memory location at address B + 2 \* content of register C to register A).

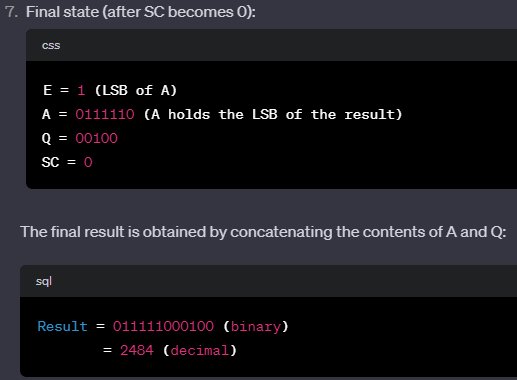
Q.15 Display the values stored in registers E, A, Q, and SC throughout the multiplication procedure of the binary numbers 11111 (multiplicand) and 10101 (multiplier). The given binary representation does not include signs.











Q.16 Specify the control word that must be applied to the processor having general register organisation

to implement the following microoperations. Define the control word necessary for a processor with a

general register organization to execute the following microoperations. Note that operation codes for

add, complement, decrement, shift left and input are 00010, 01110, 00110, 11000 and 00000

respectively.

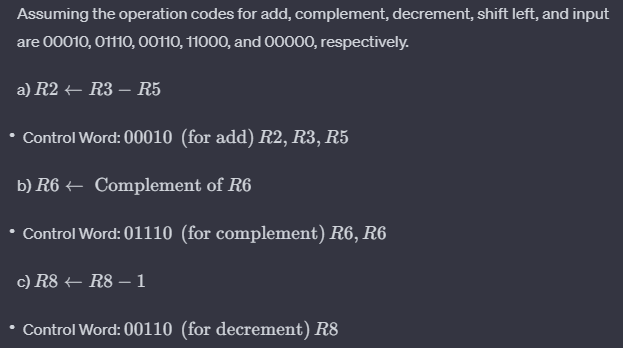
a) R2  R3 - R5

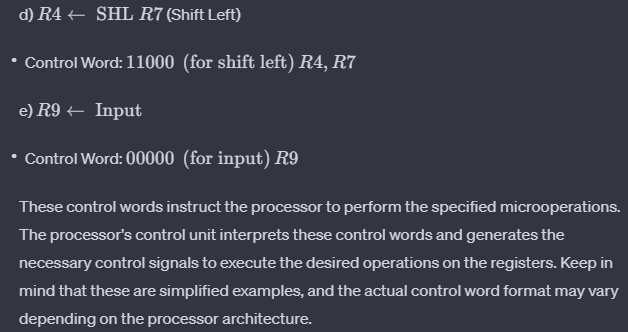
b) R6  Complement of R6

c) R8  R8 -1

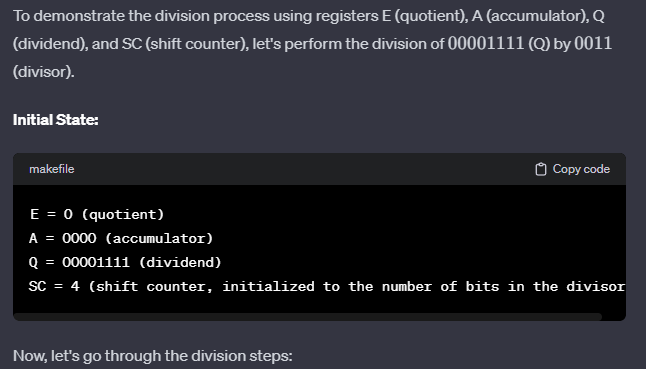
d) R4  SHL R7

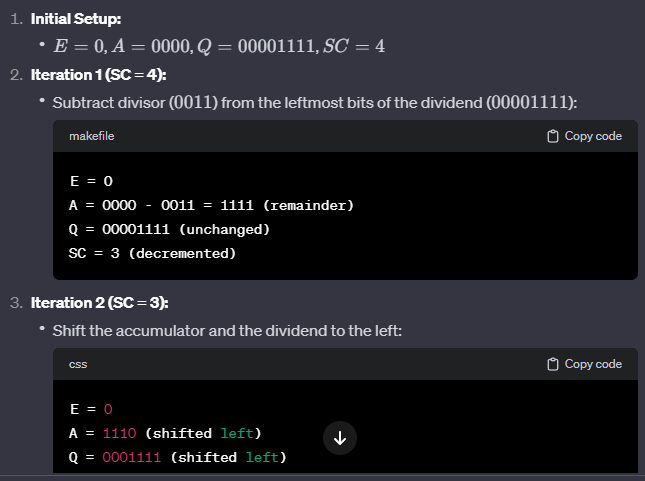
e) R9  Input

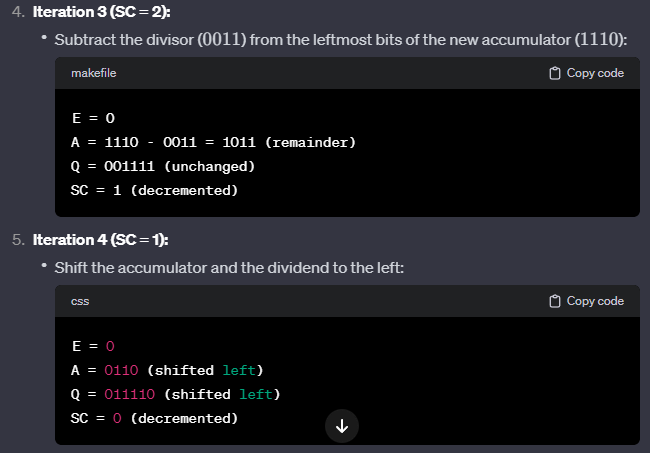


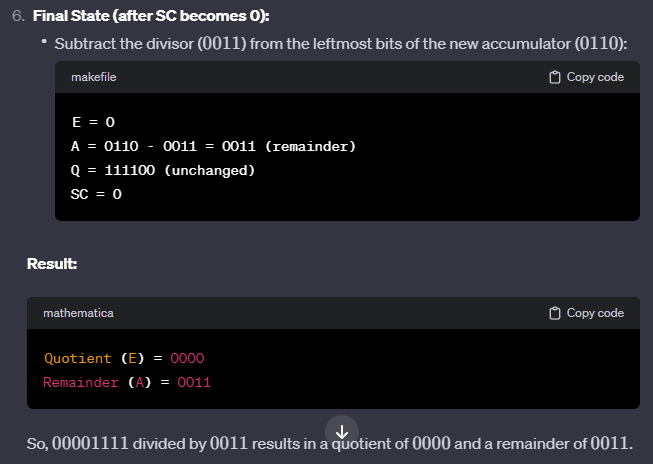


Q.17 Show the contents of registers E, A, Q, and SC during the process of division of 00001111 by 0011.

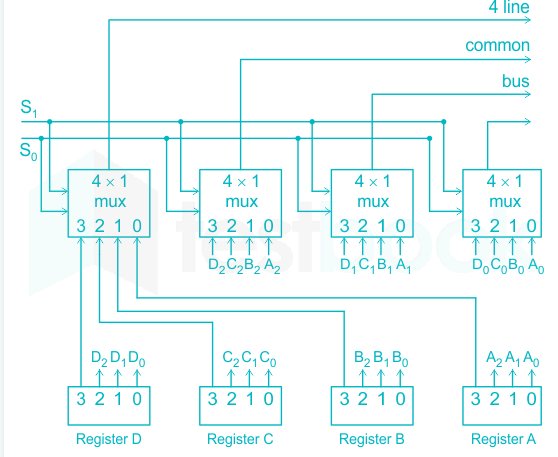




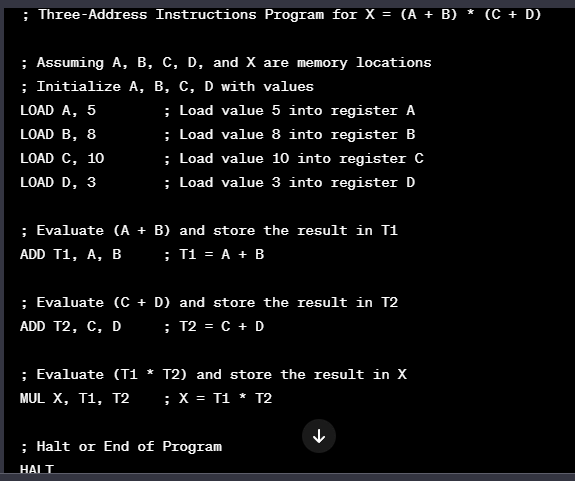




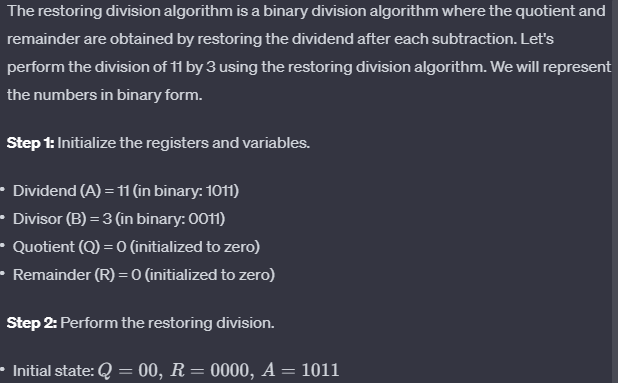
Q.18 Design a common bus system for a digital computer system having 4 registers, each of 8-bit size.

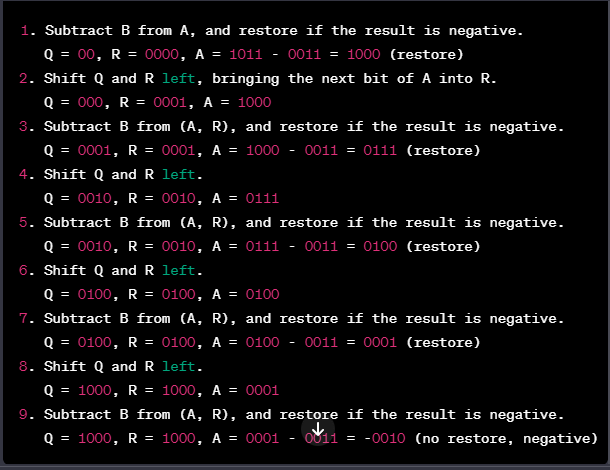


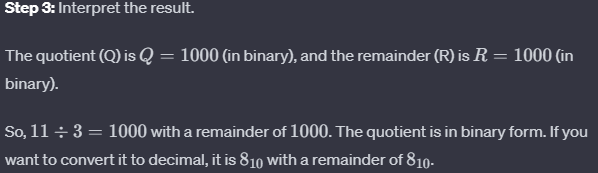
Q.19 Create a program to assess the given arithmetic expression utilizing zero, one, two, or three address instructions: X = (A +B) \* (C + D)



Q.20 Find 11 divided by 3 using restoring division algorithm.

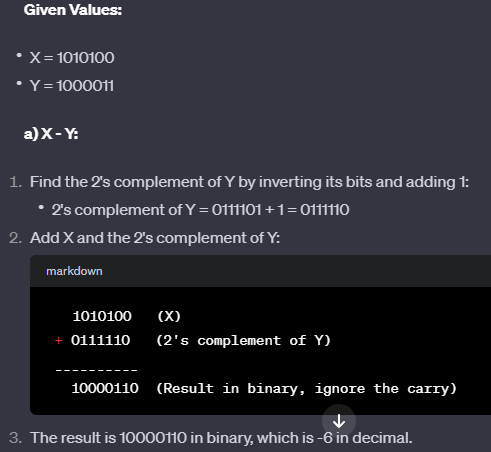


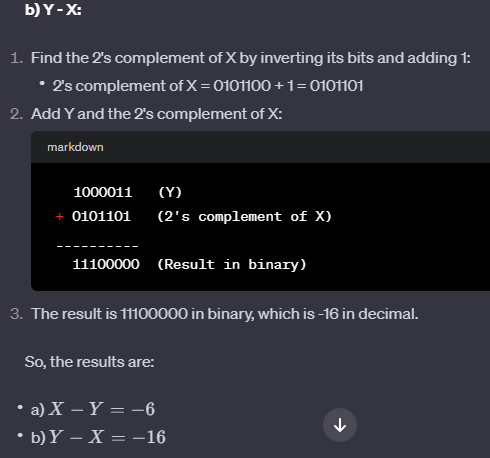




Q.21 Perform following binary subtraction using 2's complement method where the values of X and Y are 1010100 and 1000011 respectively.

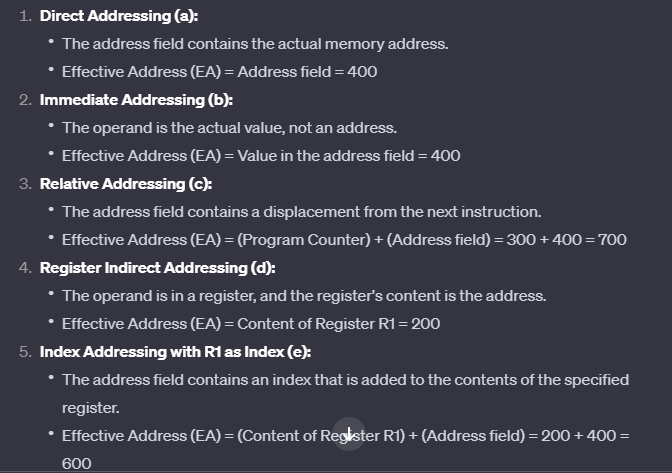
a) X – Y b) Y - X

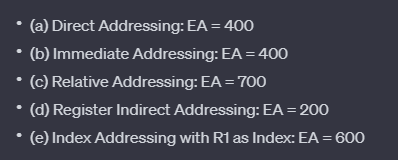




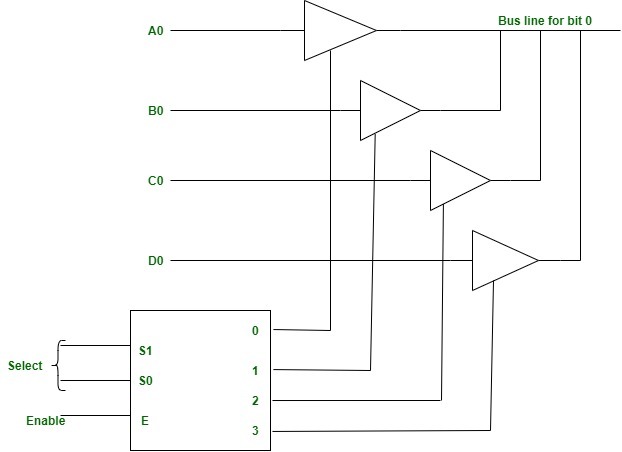
Q.22 An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is:

(a) direct; (b) immediate; (c) relative; (d) register indirect; (e) index with R1 as the index register.





Q.23 Design a common bus system for a digital computer system having four registers of 4-bit using 3- state buffers.



Q.24 A bus-organized CPU (general register organisation) has 16 registers with 32 bits in each, an ALU, and a destination decoder. Evaluate following.

(a) How many multiplexers are there in the A bus (or B bus), and what is the size of each multiplexer.

(b) How many selection inputs are needed for MUX A and MUX B?

(c) How many inputs and outputs are there in the decoder?

(d) How many inputs and outputs are there in the ALU for data, including input and output carries?

(e) Formulate a control word for the system assuming that ALU has 40 operations.

