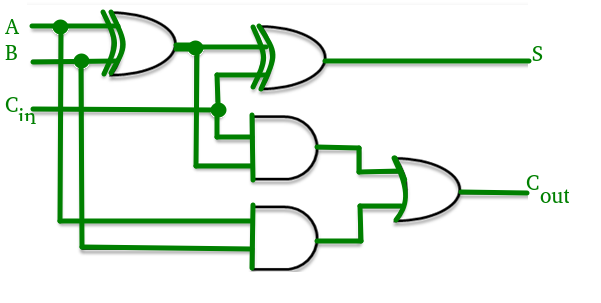
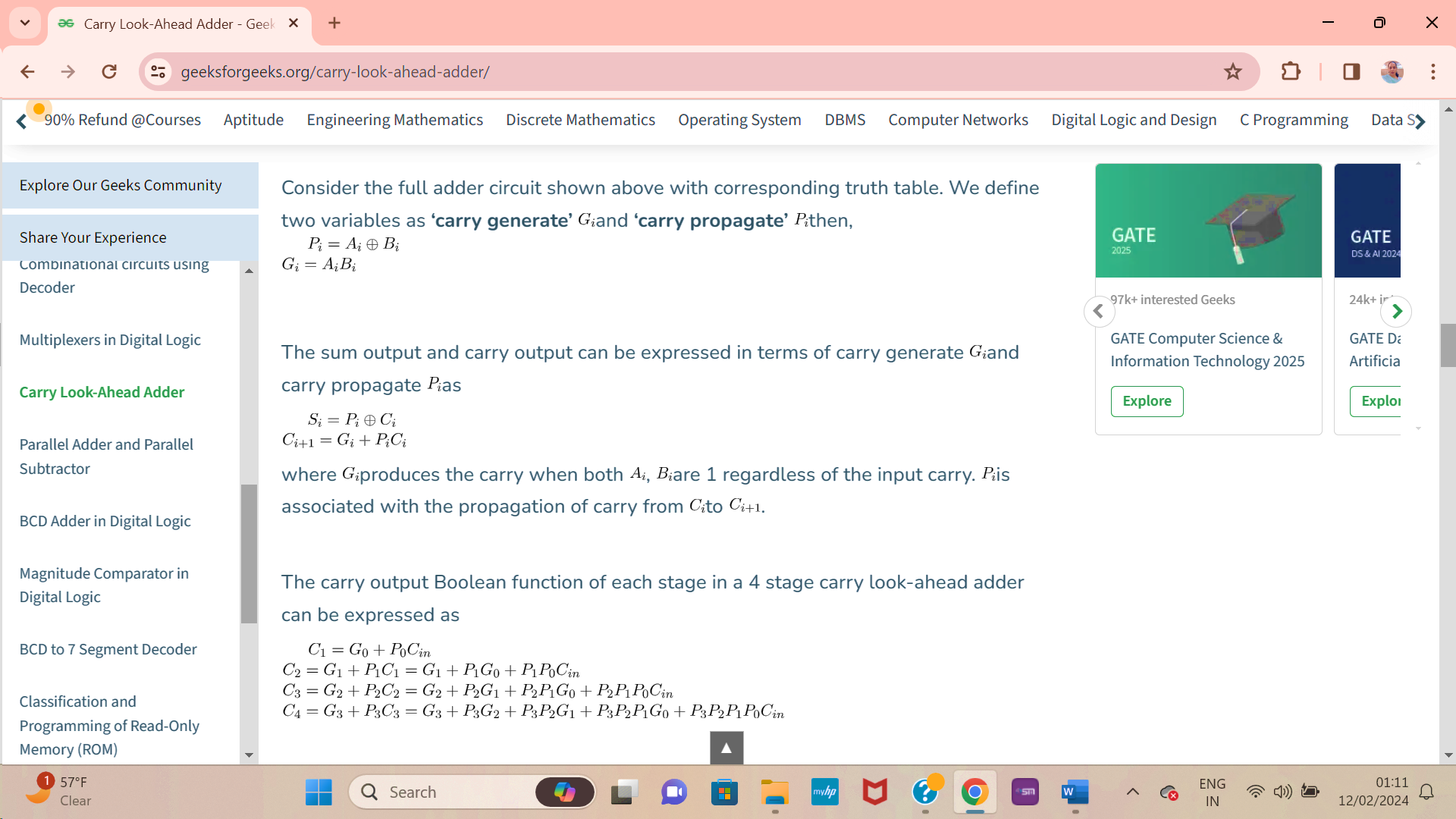
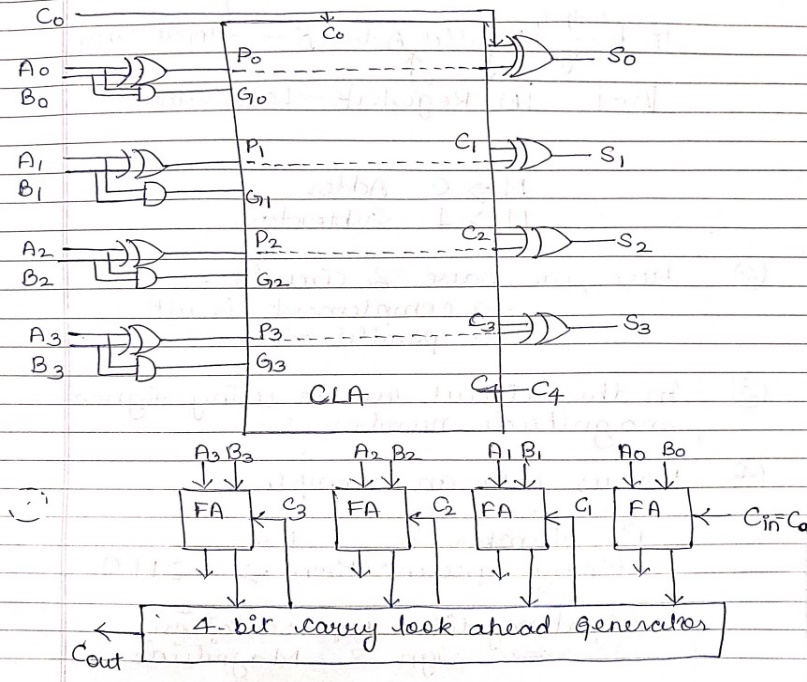
**Solution Practice Set -1 for 5th Semester COA ETE Examination**

**Q.8 Design 4-bit carry look ahead adder.**

A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic.





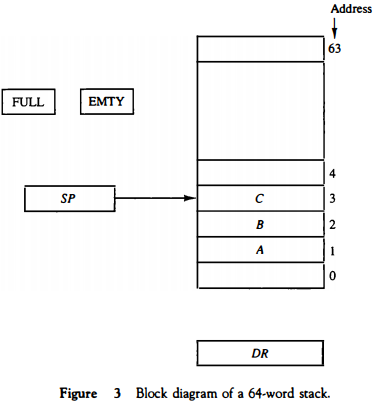


**Q.9 Explain the microoperations involved for the PUSH and POP operations in a register stack and also for a memory stack with an example.**

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved (LIFO).**The stack in digital computers** is essentially a memory unit with an address register that can count only (after an initial value is loaded into it).**The register that holds** the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack. **The two operations** of a stack are the insertion and deletion of items. However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

**Register Stack**

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure 3 shows the organization of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3.



**To remove the top item**, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. **Item B is now on top of the stack since SP holds address 2**. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed. **This does not matter because when the stack is pushed**, a new item is written in its place. In a 64-word stack, the stack pointer contains 6 bits because 26 = 64. **Since SP has only six bits**, it cannot exceed a number greater than 63 (111111 in binary). When63 is incrementedby 1, the resultis 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits. **Similarly, when 000000 is decremented by 1**, the result is 111111. The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written into or read out of the stack. **Initially**, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

**The push operation** is implemented with the following sequence of microoperations;

SP ← SP + 1 Increment stack pointer

M[SP] ← DR Write item on top of the stack

If (SP = 0) then (FULL ←1) Check if stack is full

EMTY ← 0 Mark the stack not empty

**The stack pointer** is incremented so that it points to the address of the next-higher word. A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP. **The first item stored** in the stack is at address L The last item is stored at address 0. **If SP reaches 0**, the stack is full of items, so FULL is set to L This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0. **Once an item is stored** in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0. **A new item**is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of microoperations:

DR ← M[SP] Read item from the top of stack

SP ← SP - 1 Decrement stack pointer

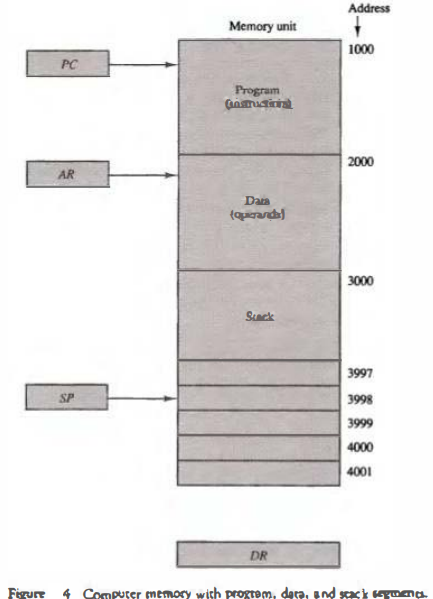
If (SP = 0) then (EMTY ← 1) Check if stack is empty

FULL ← 0 Mark the stack not full

**The top item** is read from the stack into DR . The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1. **This condition** is reached if the item read was in location 1. Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. Note that if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equivalent to decimal 63. **In this configuration**, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1.

**Memory Stack**

**A stack can exist** as a stand-alone unit as in Fig. 3 or can be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. **Figure** shows a portion of computer memory partitioned into three segments: program, data, and stack. The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data.

.

**The stack pointer**SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. **PC is used during the fetch phase**to read an instruction. AR is used during the execute phase to read an operand. **SP is used to push or pop items** into or the stack. As shown in Fig. 4, the initial value of SP is 4001 and the stack grows with decreasing addresses. **Thus, the first item** stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack Is 3000.**No provisions** are available for stack limjt checks. **We assume that the items** in the stack communicate with a data register DR . A new item is inserted with the push operation as follows:

SP ← SP - 1

M[SP] ← DR

**The stack pointer**is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

DR ← M[SP]

SP ← SP + 1

**The top item** is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack. **Most computers** do not provide hardware to check for stack overflow (full stack) or underflow (empty stack). **The stack limits** can be checked by using two processor registers: one to hold the upper limit (3000 in this case), and the other to hold the lower limit (4001 in this case). **After a push operation**, SP is compared with the upper-limit register and after a pop operation, SP is compared with the lower-limit register. **The two microoperations** needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP. Which of the two microoperations is done first and whether SP is updated by incrementing or decrementing depends on the organization of the stack **In Fig. the stack grows** by decreasing the memory address. The stack may be constructed to grow by increasing the memory address as in Fig. **In such a case, SP is incremented** for the push operation and decremented for the pop operation. A stack may be constructed so that SP points at the next empty location above the top of the stack. **In this case the sequence** of microoperations must be interchanged. A stack pointer is loaded with an initial value. This initial value must be the bottom address of an assigned stack in memory. Henceforth, SP is automatically decremented or incremented with every push or pop operation.

**The advantage of a memory stack** is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

**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?**

Given:

Instruction Size=32 bit

Operand Size=12 bit

Instruction Format for 2 Address Instruction:

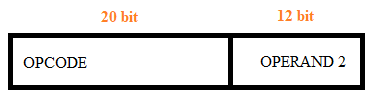


Number of two- address instructions possible= 2**8** = **256**

Number of two-Address instructions available= 250

256–250=6 two Address instructions are free which can be used indicate the given instruction is a one- Address instruction.

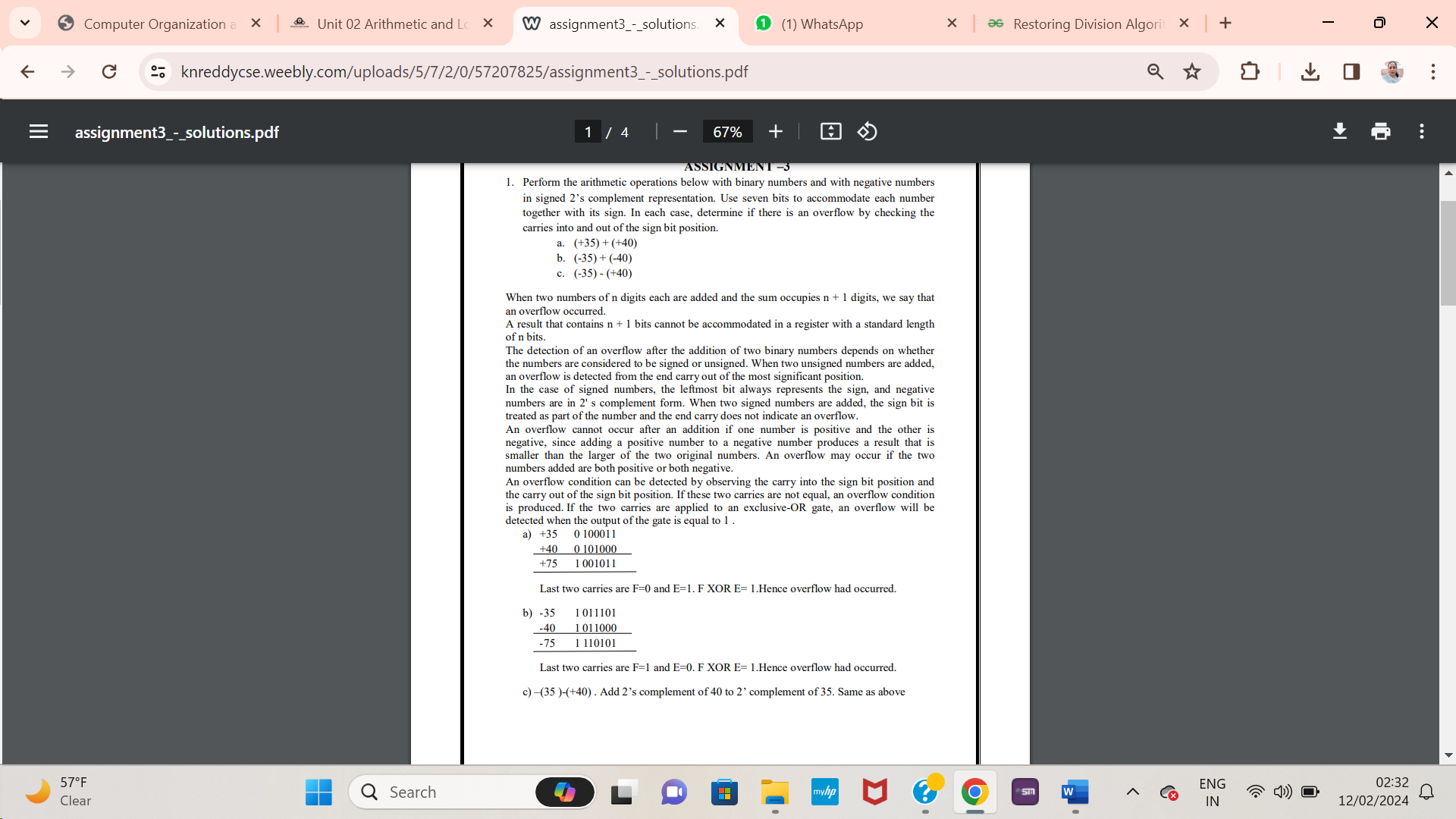
So, number of one-Address instruction= 6\*2**12** = **24576.**



The above problem has been solved using Expanding Opcode technique (i.e. length of opcode is not fixed).

**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:**

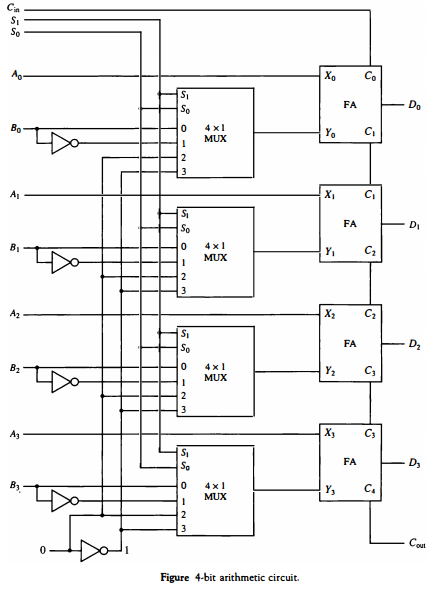
1. **(+35) + (+40) b) (-35) + (-40) c) (-35) - (+40)**

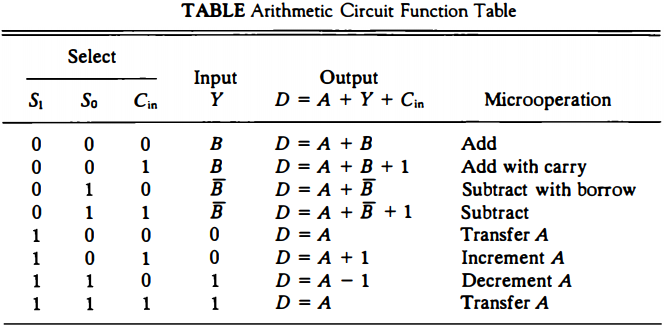


**Q.12 Design and explain the working of 4-bit Arithmetic Circuit that can perform any of the following micro-operations: addition, subtraction, increment, and decrement.**

4-bit Arithmetic Circuit: **The arithmetic microoperations** listed in Table below can be implemented in one composite arithmetic circuit. **The basic component** of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

**The diagram of a 4-bit arithmetic circuit** is shown in Fig. below. It has four full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations. **There are two 4-bit inputs A and B and a 4-bit output D**. The four inputs from A go directly to the X inputs of the binary adder. **Each of the four inputs from B** are connected to the data inputs of the multiplexers. The multiplexers data inputs also receive the complement of B. The other two data inputs are connected to logic-0 and logic-1. Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through an inverter whose input is 0. **The four multiplexers** are controlled by two selection inputs, S1 and S0. The input carry Cin goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next. **The output of the binary adder** is calculated from the following arithmetic sum: D =A + Y + Cin where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. **Cin is the input carry**, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. By controlling the value of Y with the two selection inputs S1 and S0 and making Cin equal to 0 or 1, it is possible to generate the eight arithmetic microoperations listed in Table below.





**When S1S0 = 00**, the value of B is applied to the Y inputs of the adder. If Cin = 0, the output D = A + B . If Cin = 1, output D = A + B + l. Both cases perform the add microoperation with or without adding the input carry.

**When S1S0 = 01**, the complement of B is applied to the Y inputs of the adder. If Cin = 1, then D = A + B + 1. This produces A plus the 2's complement of B, which is equivalent to a subtraction of A - B.

**When Cin = 0**, then D = A + B. This is equivalent to a subtract with borrow, that is, A - B - 1. When S1S0 = 10, the inputs from B are neglected, and instead, all O's are inserted into the Y inputs. The output becomes D = A + 0 + Cm路 This gives D = A when Cin = 0 and D = A + 1 when Cin = 1. In the first case we have a direct transfer from input A to output D.

**In the second case**, the value of A is incremented by 1. When S1S0 = 11, all 1' s are inserted into the Y inputs of the adder to produce the decrement operation D = A - 1 when Cin = 0.

**This is because** a number with all 1's is equal to the 2's complement of 1 (the 2's complement of binary 0001 is 1111). Adding a number A to the 2's complement of 1 produces F = A + 2's complement of 1 = A - 1. When Cin = 1, then D = A - 1 + 1 = A, which causes a direct transfer from input A to output D.

\*\***Note that the microoperation D = A** is generated twice, so there are only seven distinct microoperations in the arithmetic circuit.

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

1. (+14) X (-14)
2. (-7) X (+3)
3. (10010)2 X (10100)2

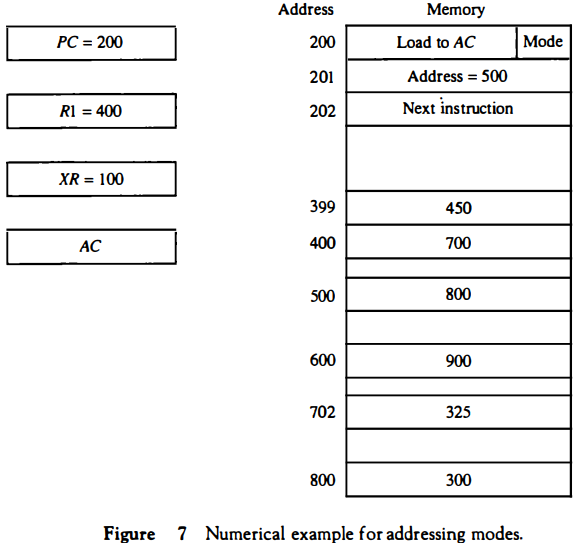
**Q.14 Explain the various addressing modes with the help of a numerical example**

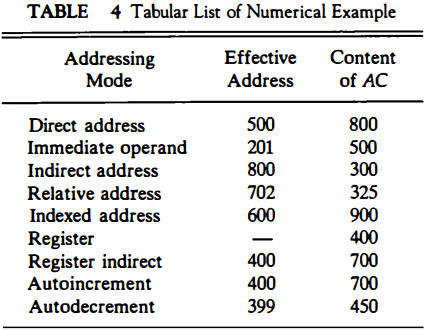
**The operation field** of an instruction specifies the operation to be performed. **This operation** must be executed on some data stored in computer registers or memory words. **The way the operands** are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

1. **Implied Mode: In this mode the operands** are specified implicitly in the definition of the instruction. **For example**, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.**Zero-address instructions** in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.
2. **Immediate Mode: In this mode the operand** is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.
3. **Register Mode: In this mode** the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. **A k-bit field** can specify any one of 2k registers.
4. **Register Indirect Mode**: **In this mode the instruction** specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself. **A reference** to the register is then equivalent to specifying a memory address.
5. **Autoincrement or Autodecrement Mode:This is similar** to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. **When the address** stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.**This can be achieved**by using the increment or decrement instruction.
6. **Direct Address Mode:** **In this mode the effective address** is equal to the address part of the instruction. **The operand resides** in memory and its address is given directly by the address field of the instruction.
7. **Indirect Address Mode: In this mode the address field** of the instruction gives the address where the effective address is stored in memory.
8. **Relative Address Mode: In this mode the content of the program counter** is added to the address part of the instruction in order to obtain the effective address.
9. **Indexed Addressing Mode: In this mode the content** of an index register is added to the address part of the instruction to obtain the effective address. **The index register** is a special CPU register that contains an index value. **The address field of the instruction** defines the beginning address of a data array in memory. **Each operand in the array** is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the index register. **Any operand in the array** can be accessed with the same instruction provided that the index register contains the correct index value.
10. **Base Register Addressing Mode: In this mode the content of a base register** is added to the address part of the instruction to obtain the effective address.**This is similar to the indexed addressing mode** except that the register is now called a base register instead of an index register. **The difference** between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction.**A base register** is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

Numerical Example

**To show the differences** between the various modes, we will show the effect of the addressing modes on the instruction defined in Fig. **The two-word instruction at address 200 and 201** is a "load to AC" instruction with an address field equal to 500. **The first word of the instruction** specifies the operation code and mode, and the second word specifies the address part. **PC has the value 200 for fetching this instruction**. The content of processor register R1 is 400, and the content of an index register XR is 100. **AC receives the operand** after the instruction is executed.





**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.**

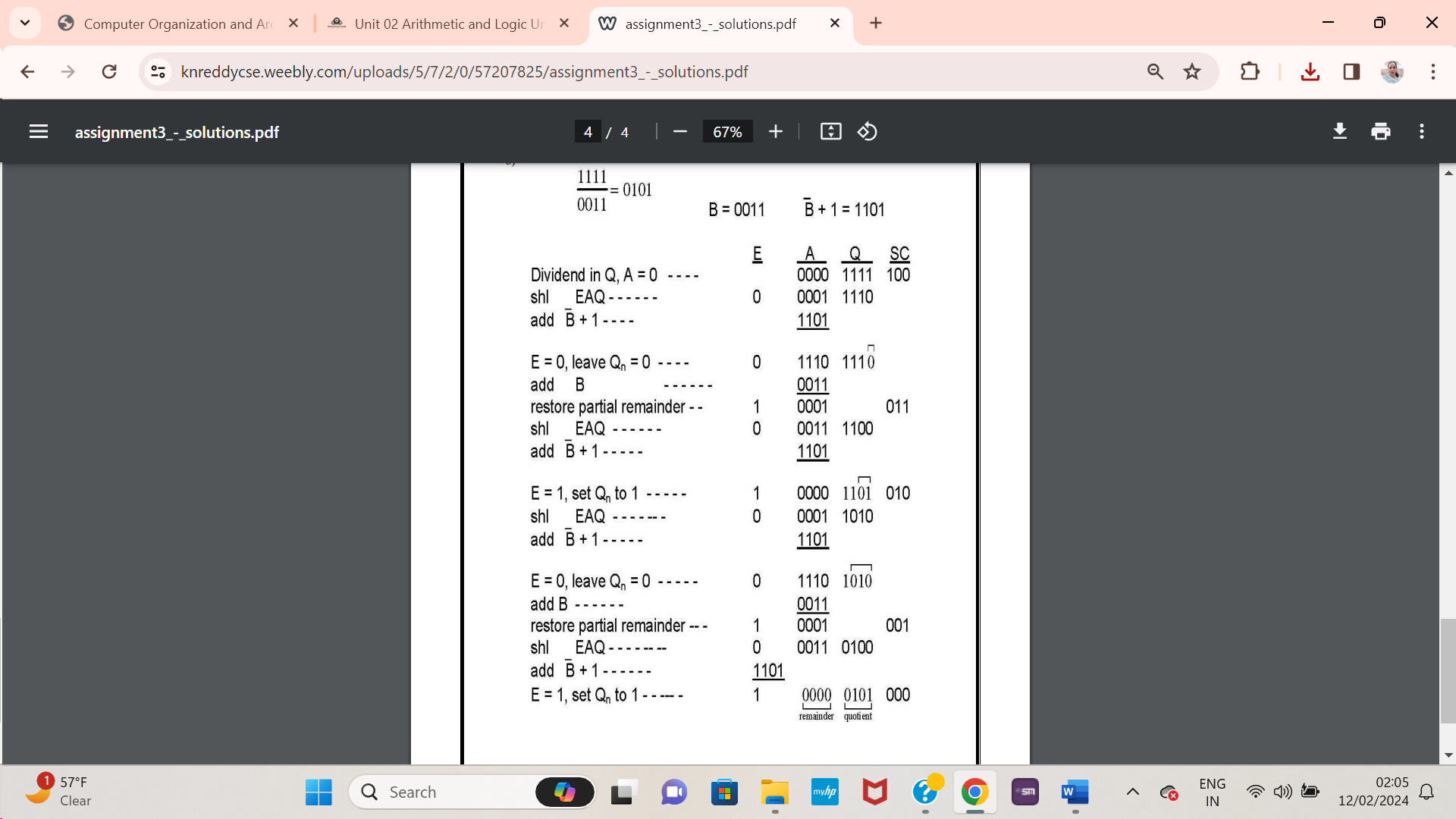


**Final Product is in AQ= 00000 00001**

**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.**

1. **R2 🡨 R3 - R5**
2. **R6 🡨 Complement of R6**
3. **R8 🡨 R8 -1**
4. **R4 🡨 SHL R7**
5. **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.**

**A bus structure** consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer.

**One way of constructing a common bus system** is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus. The construction of a bus system for four registers is shown in Fig. below. Each register has eight bits, numbered 0 through 7. **The bus consists of eight 4 x 1 multiplexers** each having four data inputs, 0 through 3, and two selection inputs, S1 and S0. In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers.

**For example**, output 1 of register A is connected to input 0 of MUX 1 because this input is labeled A1. The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus.

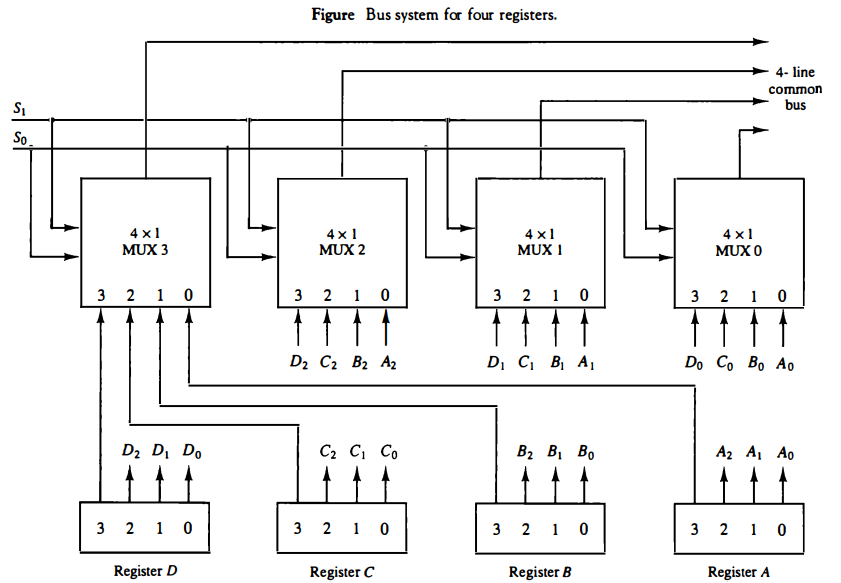
**Thus, MUX 0 multiplexes** the four 0 bits of the registers, MUX 1 multiplexes the four 1 bits of the registers, and similarly for the other six bits.

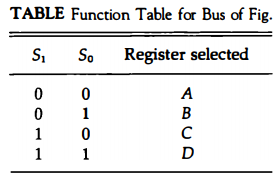
**The two selection lines S1 and S0** are connected to the selection inputs of all eight multiplexers. **The selection lines** choose the four bits of one register and transfer them into the eight-line common bus. When S1S0 = 00, the 0 data inputs of all eight multiplexers are selected and applied to the outputs that form the bus. **This causes the bus lines** to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

**Similarly, register B** is selected if S1S0 = 01, and so on. Table below shows the register that is selected by the bus for each of the four possible binary value of the selection lines

**In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus. The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register. The size of each multiplexer must be k x 1 since it multiplexes k data lines.**

NOTE: Below Diagram is for 4 registers, each of 4-bit but we have to draw for 4 registers each of 8-bit as given in question so modify this diagram by addition of 4 more MUXs and each registers having 8-bit instead of 4 as per explanation given above.





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

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organizations:

* 1. Single accumulator organization.
* 2. General register organization.
* 3. Stack organization.

**Three-Address Instructions**

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) \* (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

* ADD R1, A, B R1 ← M[A] + M[B]
* ADD R2, C, D R2 ← M[C] + M[D]
* MOL X, R1, R2 M[X] ← R1 \* R2

It is assumed that the computer has two processor registers, R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A. **The advantage** of the three-address format is that it results in short programs when evaluating arithmetic expressions. **The disadvantage** is that the binary-coded instructions require too many bits to specify three addresses.

**Two-Address Instructions**

Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) \* (C + D) is as follows:

* MOV R1, A R1 ← M[A]
* ADD R1, B R1 ← R1 + M[B]
* MOV R2, C R2 ← M[C]
* ADD R2, D R2 ← R2 + M[D]
* MOL R1, R2 R1 ← R1 \* R2
* MOV X, R1 M[X] ← R1

**The MOV instruction** moves or transfers the operands to and from memory and processor registers. **The first symbol**listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

**One-Address Instructions**

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations. **The program to evaluate** X = (A + B) \* (C + D) is

* LOAD A AC ← M[AJ
* ADD B AC ← AC + M[B]
* STORE T M[T] ← AC
* LOAD c AC ← M[C]
* ADD D AC ← AC + M[D]
* MOL T AC ← AC\*M[T]
* STORE X M[X] ← AC

**All operations are done** between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

**Zero-Address Instructions**

**A stack-organized computer** does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) \* (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

* PUSH A TOS ← A
* PUSH B TOS ← B
* ADD TOS ← (A + B)
* PUSH C TOS ← C
* PUSH D TOS ← D
* ADD TO S ← (C + D)
* MOL TOS ← (C + D)\*(A + B)
* POP X M[X] ← TOS

**To evaluate arithmetic expressions** in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

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

Division using Restoring Algorithm where Dividend = 11, Divisor = 3

| **n** | **M** | **A** | **Q\_** | **Operation** |
| --- | --- | --- | --- | --- |
| 4 | 00011 | 00000 | 1011 | initialize |
|  | 00011 | 00001 | 011\_ | shift left AQ |
|  | 00011 | 11110 | 011\_ | A=A-M |
|  | 00011 | 00001 | 0110 | Q[0]=0 And restore A |
| 3 | 00011 | 00010 | 110\_ | shift left AQ |
|  | 00011 | 11111 | 110\_ | A=A-M |
|  | 00011 | 00010 | 1100 | Q[0]=0 |
| 2 | 00011 | 00101 | 100\_ | shift left AQ |
|  | 00011 | 00010 | 100\_ | A=A-M |
|  | 00011 | 00010 | 1001 | Q[0]=1 |
| 1 | 00011 | 00101 | 001\_ | shift left AQ |
|  | 00011 | 00010 | 001\_ | A=A-M |
|  | 00011 | 00010 | 0011 | Q[0]=1 |

Remember to restore the value of A most significant bit of A is 1. As that register Q contain the quotient, i.e. 3 and register A contain remainder 2.

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

1. 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**.

a) Direct address mode: address field value is the Effective Address (EA)= **400**

b) Immediate address mode: Instruction itself has operand in Address field. So, EA is **301**

c) Relative mode: EA = Address field + Program Counter(PC) =

So, 400+302 = **702**

d) Register Indirect Addr. Mode: EA = content of particular register (R1) = **200**

e) Index Address Mode: Used with Array handling

EA = Address Field + Index register value

So, 400+200 = **600**

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

**A bus system** can be constructed with three-state gates instead of multiplexers. A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate. **The third state** is a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance. **Three-state gates** may perform any conventional logic, such as AND or NAND. However, the one most commonly used in the design of a bus system is the buffer gate. **The graphic symbol** of a three-state buffer gate is shown in Fig. below. It is distinguished from a normal buffer by having both a normal input and a control input. The control input determines the output state.

**When the control input** is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input.

**When the control input** is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.

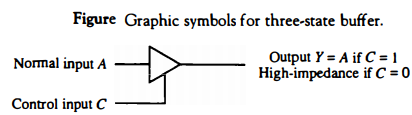
**The high-impedance state** of a three-state gate provides a special feature not available in other gates. Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.

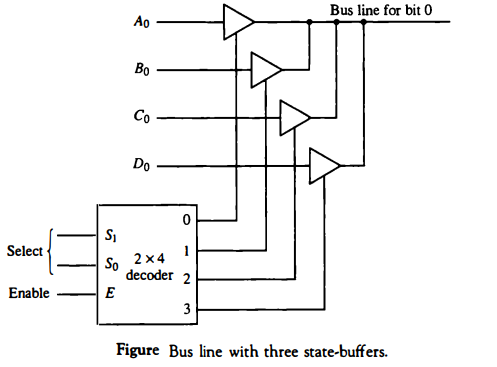
**The construction of a bus system** with three-state buffers is demonstrated in Fig. below. The outputs of four buffers are connected together to form a single bus line.

**No more than one buffer** may be in the active state at any given time. The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high impedance state.

**One way to ensure** that no more than one control input is active at any given time is to use a decoder, as shown in the diagram. When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.

**When the enable input is active**, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder. Careful investigation will reveal that Fig. below is another way of constructing a 4 x 1 multiplexer since the circuit can replace the multiplexer in Fig. Bus system for four registers.**To construct a common bus** for four registers of n bits each using three **state buffers**, we need n circuits with four buffers in each as shown in Fig. above. Each group of four buffers receives one significant bit from the four registers. **Each common output** produces one of the lines for the common bus for a total of n lines. Only one decoder is necessary to select between the four registers.





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.