ASSIGNMENT-01

COMPUTER ORGANISATION

Kaustikey Gaurav
18010304S
CSE-BDA

Q-1- Explain different instruction format with example

Answer → The most common field found in the instruction format are:

i). Operation Code field that specifies the operation to be performed.
ii) Address Field that specifies a memory address
iii). Mode Field that specifies the way that operand or effective address is determined.

We will evaluate the following arithmatic statement to show the no. of addresses. $X = (A+B) * (C+D)$

1). 3 Address Instruction → In this format we use each address field to specify either a processor register or memory operand. The program in assembly language to evaluate the given expression is as follows:

```
Add   R1, A, B    R1 ← M[A] + M[B] // Content of A = M[A]
Add   R2, C, D    R2 ← M[C] + M[D]
Mul   X, R1, R2   M[X] ← R1 * R2.
```

It is assumed that computer has 2 processor register R_1 & R_2 & symbol 'M[A]' denote the content of operand at memory address A.

2). Two Address Instruction → The 2 address instruction are most common in commercial computers.

The 'MOV' instruction moves or transfer the operand too & from memory and processor register.

```
MOV   R1, A    R1 ← M[A]
ADD   R1, B    R1 ← R1 + M[B]
MOV   R2, C    R2 ← R2 + M[C]
ADD   R2, D    R2 ← R2 + M[D]
MUL   R1, R2   R1 ← R1 * R2
MOV   X, R1    M(X) ← R1
```

3). One Address Instruction → This format use an implied accumulator register for all the data manipulation. For multiplication and division. There is a need of 2nd register. All the operations are done b/w AC register & memory operand. 'T' is the address of Temporary Memory location required for storing the intermediate result.

LOAD	A	AC ← M[A]	// Loading content of A on AC
ADD	B	AC ← AC + M[B]	// Updating (Adding)
STORE	T	M[T] ← AC	
LOAD	C	AC ← M[C]	
ADD	D	AC ← AC + M[D]	
MUL	T	AC ← AC * M[T]	
STORE	X	M[X] ← AC	

4). Zero Address Instruction → A stack organized computer for the instruction addition & multiplication

The 'PUSH' & 'POP' instruction need an address field to specify the operand that communicate with the stack. The following shown how the given expression will be written for stack organized computer. Here 'TOS' shows the 'Top of the Stack'.

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← A+B
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← C+D
MUL		TOS ← (A+B) * (C+D)
POP	X	M[X] ← TOS

Q-2 → What do you mean by Addressing Modes? Explain Different addressing modes with suitable example.

Answer → The different ways of specifying the location of an operand in an instruction are called as addressing modes.

In computer architecture, there are following type of addressing modes:-

1). Implied Addressing Mode ⇒ The definition of the instruction itself specify the operands implicitly.

⇒ It is also called as Implicit Addressing Mode.

Example → In stack organized computer, Zero Address Instruction are implied mode instructions.

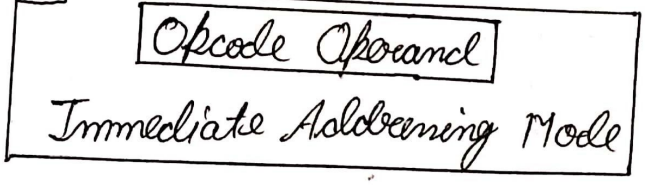
2). Stack Addressing Mode ⇒ The operand is contained at the top of the stack.

Example → ADD

- The instruction simply pops out 2 symbols contained at the top of the stack.
- The addition of these 2 operands is performed.
- The result so obtained after addition is pushed again at TOS.

3). Immediate Addressing Mode ⇒ The operand is specified in the instruction explicitly.

⇒ Instead of address field, an operand field is present that contains the operand.

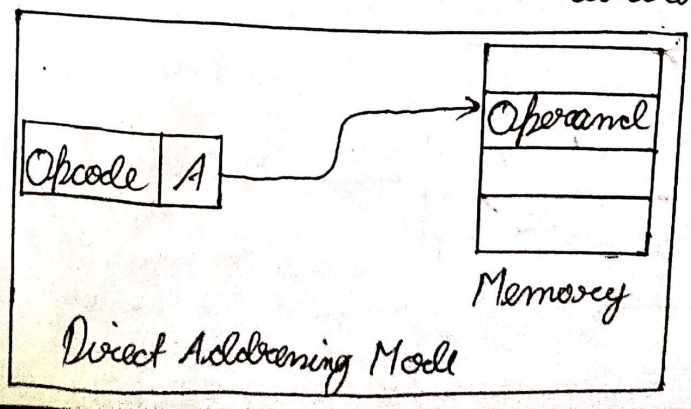


Example → MOV R #20 initializes register R to a constant value 20.

4). Direct Addressing Mode ⇒ The address field of the instruction contains the effective address of operand.

⇒ Only 1 reference to memory is required to fetch the operand.

⇒ It is also called as absolute addressing mode.

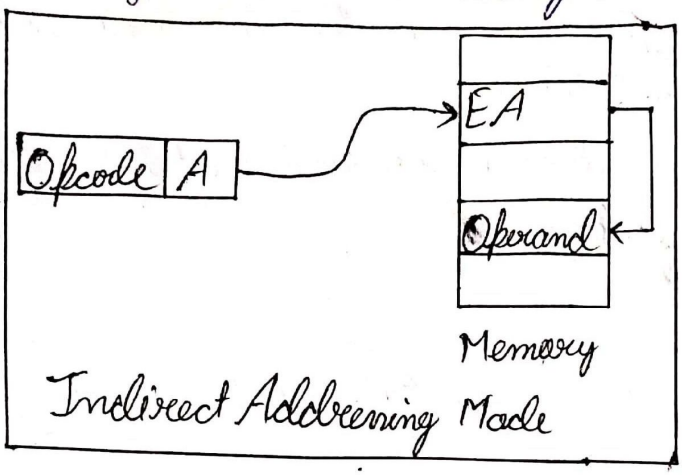


Example → ADD X will increment the value stored in the accumulator by the value stored at memory location X.

AC ← AC + [X]

5). Indirect Addressing Mode ⇒ The address field of the instruction specifies the address of memory loc^{n} that contains the effective address of the operand.

⇒ 2 references to memory are required to fetch the operand.



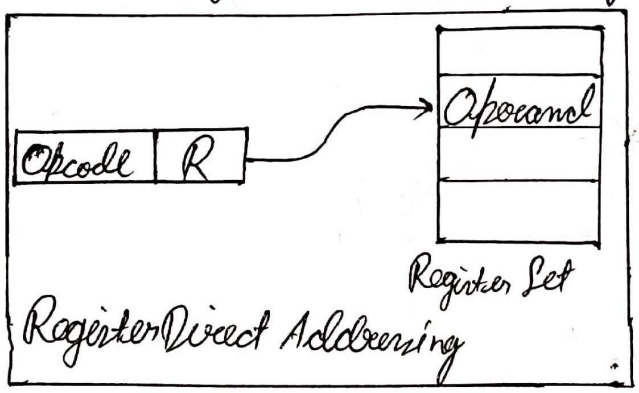
Example → ADD X will increment the value stored in the accumulator by the value stored at memory location specified by X.

$AC \leftarrow AC + [[X]]$

6). Register Direct Addressing Mode ⇒ The operand is contained in a register set.

⇒ The address field of the instruction refers to a CPU register that contains the operand.

⇒ No reference to memory is required to fetch the operand.



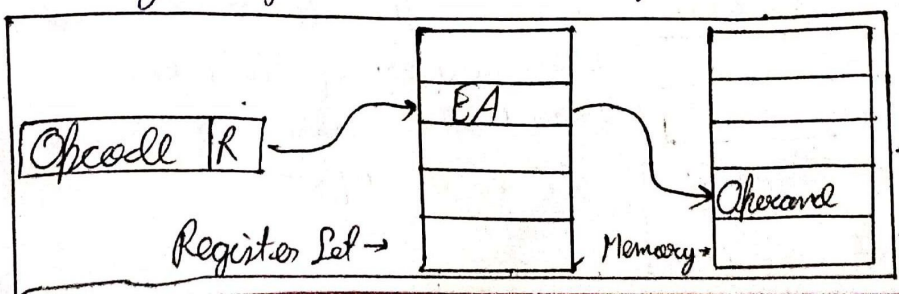
Example → ADD R will increment the value stored in the accumulator by the content of register R.

$AC \leftarrow AC + [R]$

☆ → This addressing mode is similar to direct addressing mode. The only difference is address field of the instruction refers to a CPU register instead of main memory. Similarly for indirect one in that Instruction → CPU

7). Register Indirect Addressing Mode ⇒ The address field of the instruction refers to a CPU register that contains the effective address of the operand.

⇒ Only 1 reference to memory is required to fetch the operand.

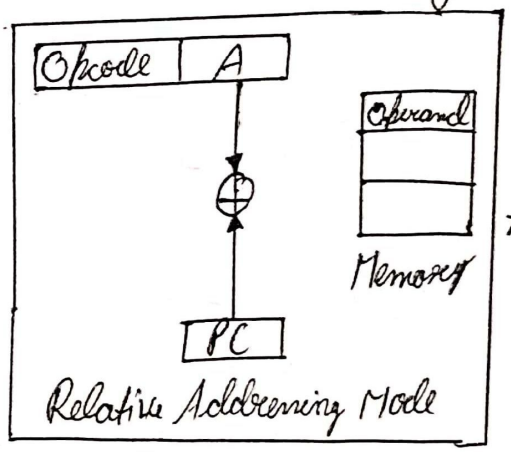


Example → ADD R will increment the value stored in the accumulator by the content of memory location specified in Register R.

$AC \leftarrow AC + [[R]]$

8). Relative Addressing Mode ⇒ Effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

Effective Address = Content of Program Counter + Address part of the instruction.



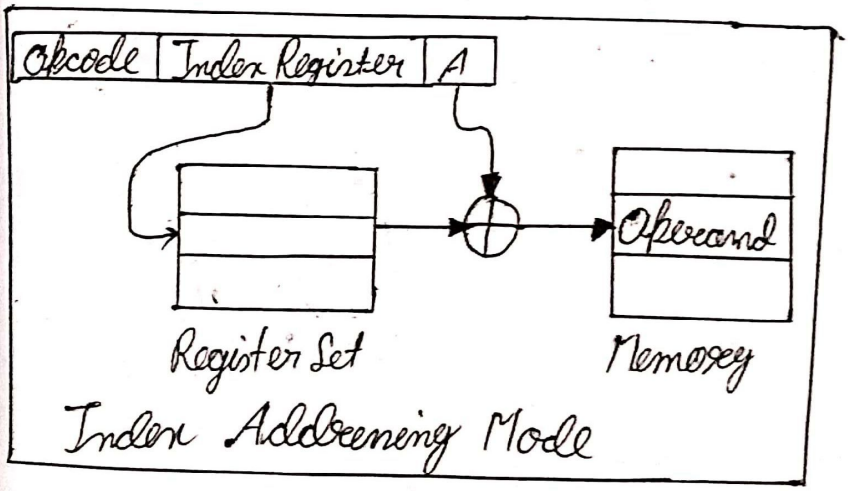
Relative Addressing Mode

☆ Program Counter (PC) always contains the address of the next instruction to be executed.

☆ After fetching the address of the instruction, the value of program counter immediately increases irrespective of whether the fetched instruction has completely executed or not.

9). Indexed Addressing Mode => Effective address of the operand is obtained by adding the content of index register with the address part of the instruction.

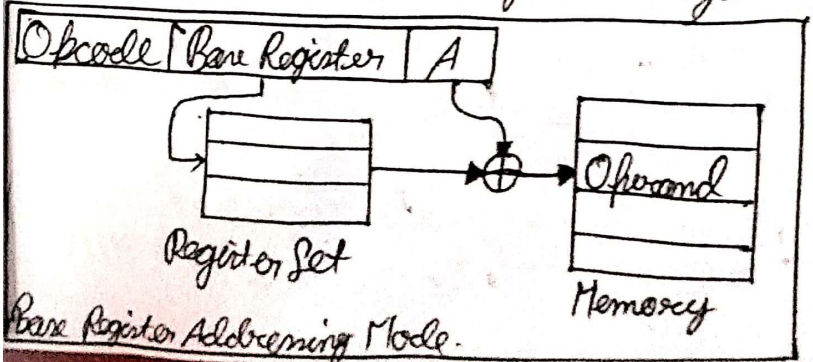
Effective Address = Content of Index Register + Address part of the Instruction.



Index Addressing Mode

10). Base Register Addressing Mode => Effective address of operand is obtained by adding the content of base register with the address part of the instruction.

Effective Address = Content of Base Register + Address Part of the instruction.



Base Register Addressing Mode.

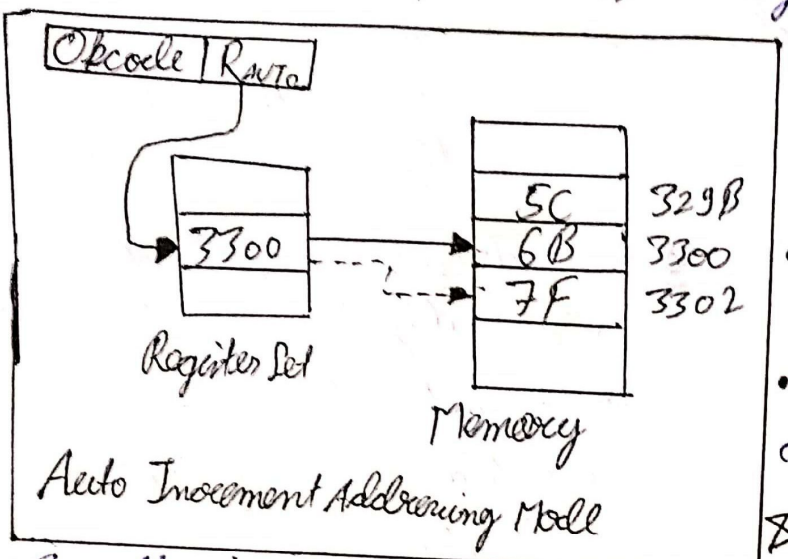
11). Auto-Increment Addressing Mode ⇒ This addressing mode is a special case of Register Indirect Addressing Mode where - Effect Address of Operand = Cont of Register.

Here ⇒ After accessing operand, the content of register is automatically incremented by step size 'd'.

⇒ Step size 'd' depends on the size of operand accessed.

⇒ Only 1 reference to memory is required to fetch the operand.

Example - Assume operand size = 2 bytes. Here,



Auto Increment Addressing Mode

- After fetching the operand 6B, the instruction register R_{AUTO} will be automatically incremented by 2.
- Then, updated value of R_{AUTO} will be 3300+2 = 3302
- At memory address 3302, the next operand will be found.

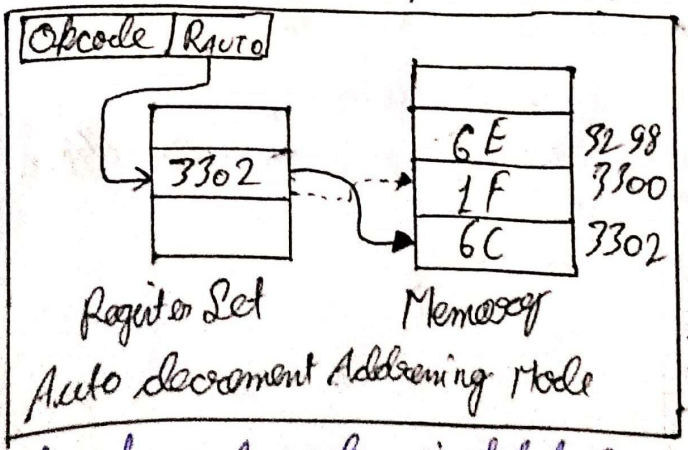
☆ - 1st the operand value is fetched. Then, the instruction register R_{AUTO} value is incremented by step size 'd'.

12). Auto-Decrement Addressing Mode ⇒ Special case of Register Indirect Addressing mode.

Effective Address of the operand = Content of Register - Step Size

- 1st the content of register is decremented by step size 'd', this depends on the size of operand accessed.
- After decrementing, the operand is read.
- Only 1 reference to memory is required to fetch the operand.

Example - Assume operand size = 2 bytes. Here,



Auto decrement Addressing Mode

- 1st, the instruction register R_{AUTO} will be decremented by 2.
- Then, updated value of R_{AUTO} will be 3302 - 2 = 3300.
- At memory address 3300 the operand will be found.

☆ - 1st, the instruction register R_{AUTO} value is decremented by step size 'd'. Then the operand value is fetched.

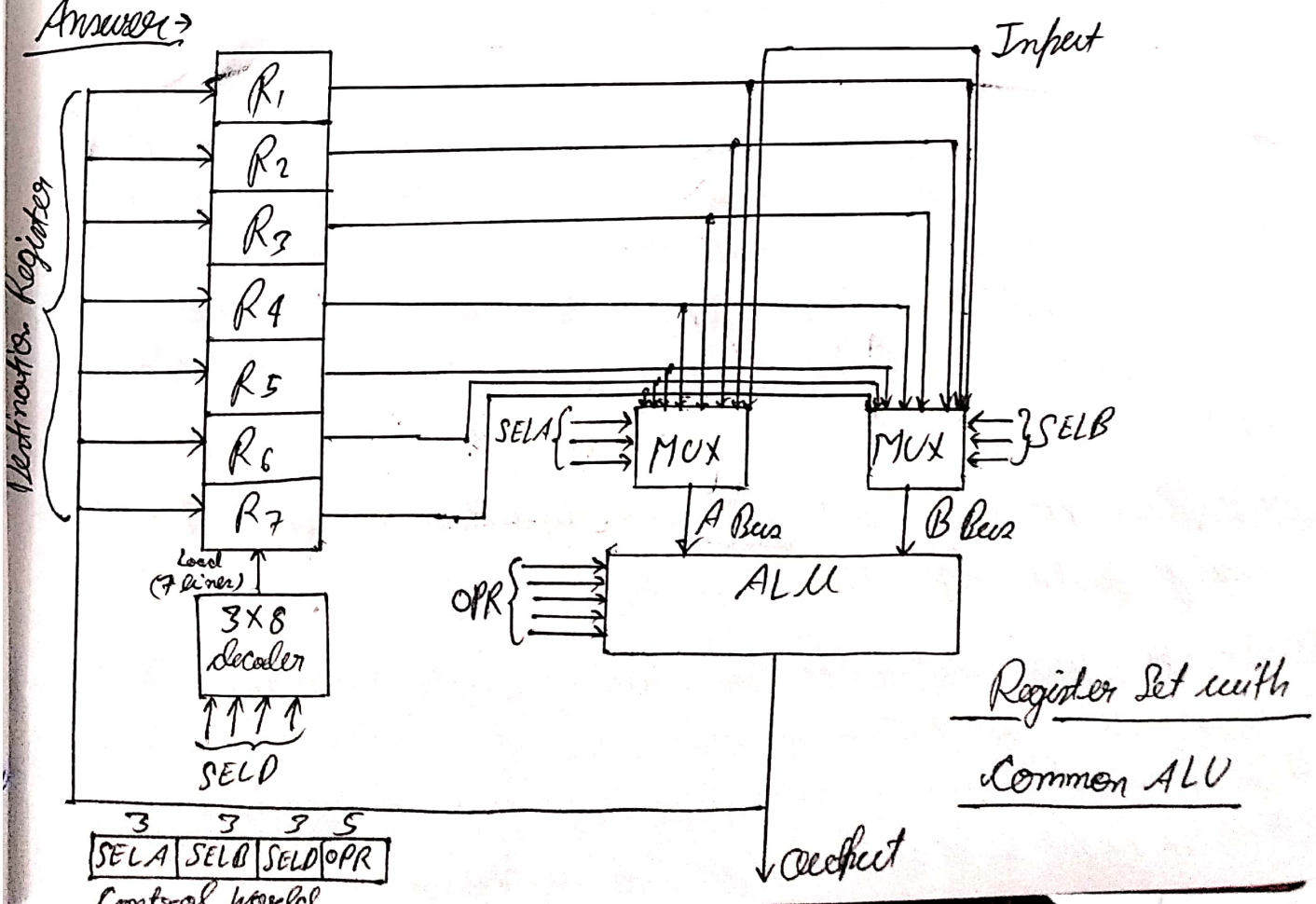
Q-3 → What do you mean by RISC? Differentiate b/w RISC & CISC.

Answer → A Reduced Instruction Set Computer is a computer which only uses simple commands that can be divided into several instructions which achieve low-level operation within a single clk cycle, as its name propose 'Reduced Instruction Set'.

CISC	RISC
1). Emphasis on hardware	Emphasis on software.
2). Multiple instruction sizes, formats	Instruction of same set with few formats
3). Less Registers	Uses more registers.
4). More addressing modes	Fewer addressing modes.
5). Extensive use of microprogramming	Complexity in compiler.
6). Instruction take a varying amount of cycle time	Instruction take one cycle time.
7). Pipelining is difficult	Pipelining is easy.

Q-4 → Explain the concept of General Register Organisation.

Answer →



Register Set with Common ALU

Encoding of Register with selection field

Binary Code	SELA	SELB	SELD
000	Input	Input	Input
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

→ External Input.

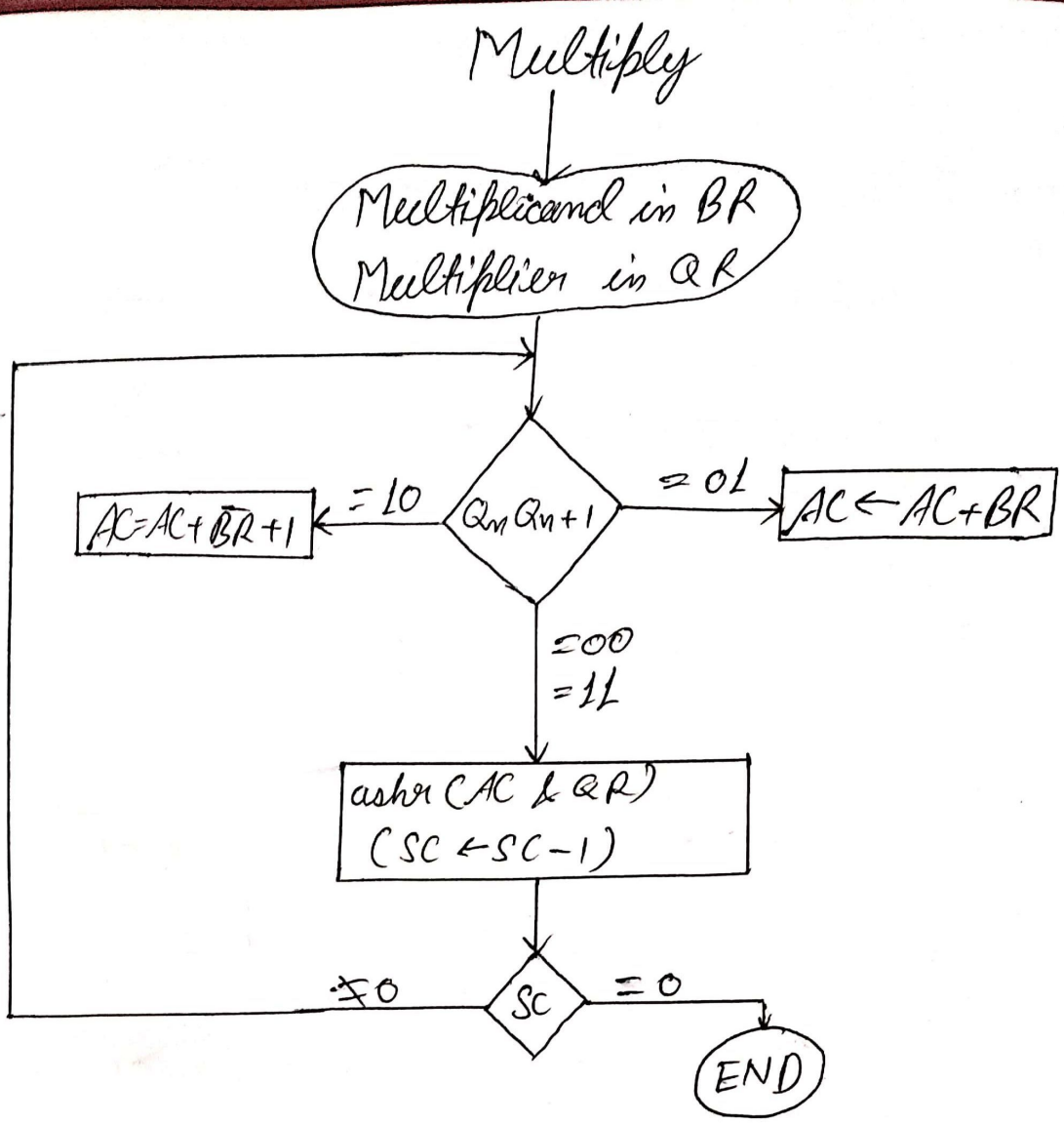
Encoding of ALU Operations

Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_i		
0	0	0	0	0	F=A	Transfer A
0	0	0	0	1	F=A+1	Increment A
0	0	0	1	0	F=A+B	Addition
0	0	0	1	1	F=A+B+1	Addition with carry
0	0	1	0	0	F=A+$\overline{B}$	Subtract with borrow
0	0	1	0	1	F=A+$\overline{B}$+1	Subtract
0	0	1	1	0	F=A-1	Decrement
0	0	1	1	1	F=A	Transfer A
0	1	0	0	0	F=A∧B	AND
0	1	0	1	0	F=A∨B	OR
0	1	1	0	0	F=A⊕B	XOR
0	1	1	1	0	F=$\overline{A}$	Complement
1	0	0	0	0	F=shr A	Shift right A to F
1	1	0	0	0	F=shl A	Shift left A into F

Based on SEL A & SEL B & Input, MUX will take the inputs from R_1 to R_7 and this will be transferred to ALU where the calculation will be calculated based on certain operation and then finally the result will be sent to output and Destination Register (In destination Register in this way it will receive new values which will be further analyse by MUX & also ALU and then again sent to register & (starting from 1)

Q-5→ Explain the flowchart of Booth Algorithm & Solve -4 x -5 using Booth Algorithm.

Answer - The figure shows the flow chart of Booth Algorithm. Initially AC & Q_{n+1} are cleared 0 & the sequence counter SC set to the no. of bits in the multiplier. The arithmetic shift right operation. Shift AC & QR to the right & leaves the sign bit unchanged.



Q_n designates the least significant bit of the meultiplier in register QR. An extra flip-flop Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.

1. Multiplier and Meultiplicand are placed in the QR & BR register Respectively.
2. Results for this will be stored in the AC & QR registers.
3. Initially AC & Q_{n+1} register will be '0'.
4. Meultiplication of a number is done in a cycle.
5. A 1-bit register Q_{n+1} is placed right of the least significant bit Q_n of the register QR.
6. In each of the cycle QR & Q_{n+1} bits will be checked.

i). If Q_n & Q_{n+1} are '11' or '00' then the bits of AC, QN & Q_{n+1} are shifted to right by 1 bit.

$-4 = 0100 \xrightarrow{1'} 1011 \xrightarrow{2'} 1100 \rightarrow 1100 \rightarrow 11100$ (Multiplicand) BR

$+1 \rightarrow 00100 \rightarrow \overline{BR}+1$

$-5 = 0101 \xrightarrow{1'} 1010 \xrightarrow{2'} 1011 \rightarrow 11011 \Rightarrow$ (Multiplier) $\rightarrow$ QR

Q_n Q_{n+1}	BR = 11100 $\overline{BR}+1$ = 00100	AC	QR	Q_{n+1}	SC
10	Initial Subtract BR	00000 00100 00100	11011	1	101
11	ashr ashr	00010	01101	1	100
01	Add BR ashr	00001 11100 11101	00110	0	011
10	Subtract BR ashr	11110 00100 00010	10011	0	010
11	ashr	00001	01001	1	001
		00000	10100	1	000

$(0000010100)_2$ = 20 Answer

Sign bit remain unchanged

ii). If the value is shown '01' then multiplicand (BR) is added to AC. After addition AC, QN, Q_{n+1} register are shifted to the right by 1 bit.

iii). If the value is shown '10' then multiplicand is subtracted ($\overline{BR}+1$) from AC. After subtraction AC, QN, Q_{n+1} register is shifted to right by 1-bit.

Then this procedure is checked again with each time a bit decrease in ~~Sec~~ Sequence Counter (SC) until it reaches from No. of bit to '0' bit.

Finally AC & QR are kept together to form complete double its original bits, which is finally converted to decimal.

Basically, Booth's Algorithm uses the concept of an arithmetic right shift in which the leftmost bit is not only shifted right by 1 bit but it also remains in the original position.