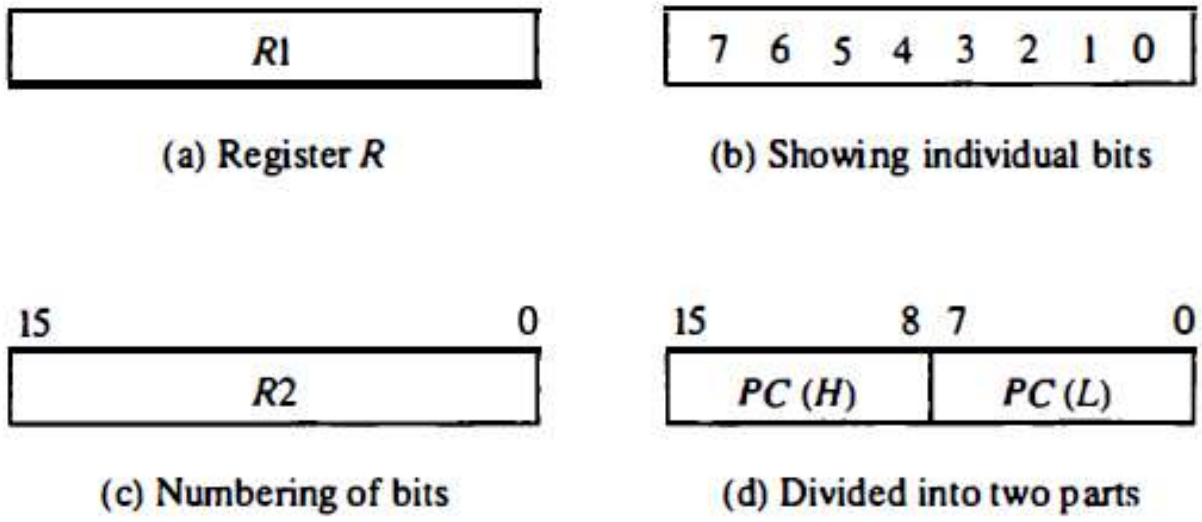
COMPUTER ORGANIZATION
MODULE-I (RTL, BOOTH MULTIPLIER & GRO)

Register, Bus and Memory Transfer

Register Transfer Language:

- ✓ The operations executed on data stored in registers are called microoperations.
- ✓ The symbolic notation used to describe the microoperation transfers among registers is called Register transfer language.
- ✓ The term Register transfer implies the availability of hardware logic circuits to perform stated microoperation.
- ✓ Computer registers are designated by capital letters to denote the function of register.



Block Diagram of Register

Register Transfer: The information transfer from one register to another register is designated in symbolic form.

For example: R2 ←R1 denotes a transfer of the content of register R1 into register R2.

The above Register transfer under a predetermined control condition can be represented by an if-then statement :

If (P=1) Then R2 ←R1

where P is a control signal generated in the control section.

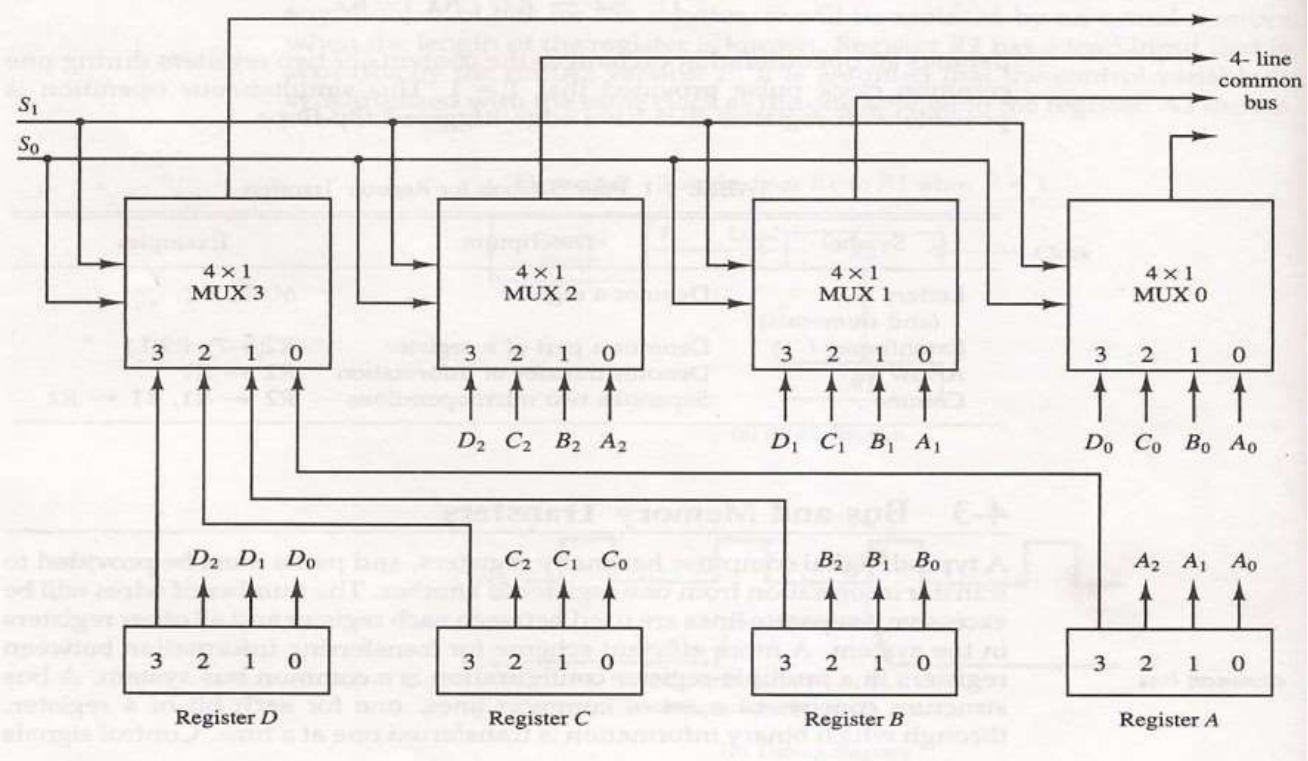
It can be written as P: R2 ←R1

Types of Registers:

Instruction register (IR)
Program counter (PC)
General-purpose register (R0 – Rn-1)
Memory address register (MAR)
Memory data register (MDR)

Bus Transfer:

- A group of lines that serves as a connecting path for several devices is called a bus.
- The efficient way for transferring information between registers in a multiple-register configuration is a common system bus.
- One way of constructing a common bus system is with multiplexers which is shown in fig.



Function Table for Bus

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Bus Transfer:

- The symbolic statement for a bus transfer may mention the bus or its presence may be implied in the statement.
- When the bus is included in the statement, the register transfer is symbolized as follows:

  BUS ← C, R1 ← BUS

- The content of register C is placed on the bus and the content of the bus is loaded into register R1 by activating its load control input.
- If the bus is known to exist in the system, it may be convenient just to show the direct transfer.

  R1 ← C

Three-state Bus Buffers:

- Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate.
- The third stage is a high-impedance state which behaves like an open circuit.
- The most commonly used in the design of a bus system is the buffer gate.

Problem:

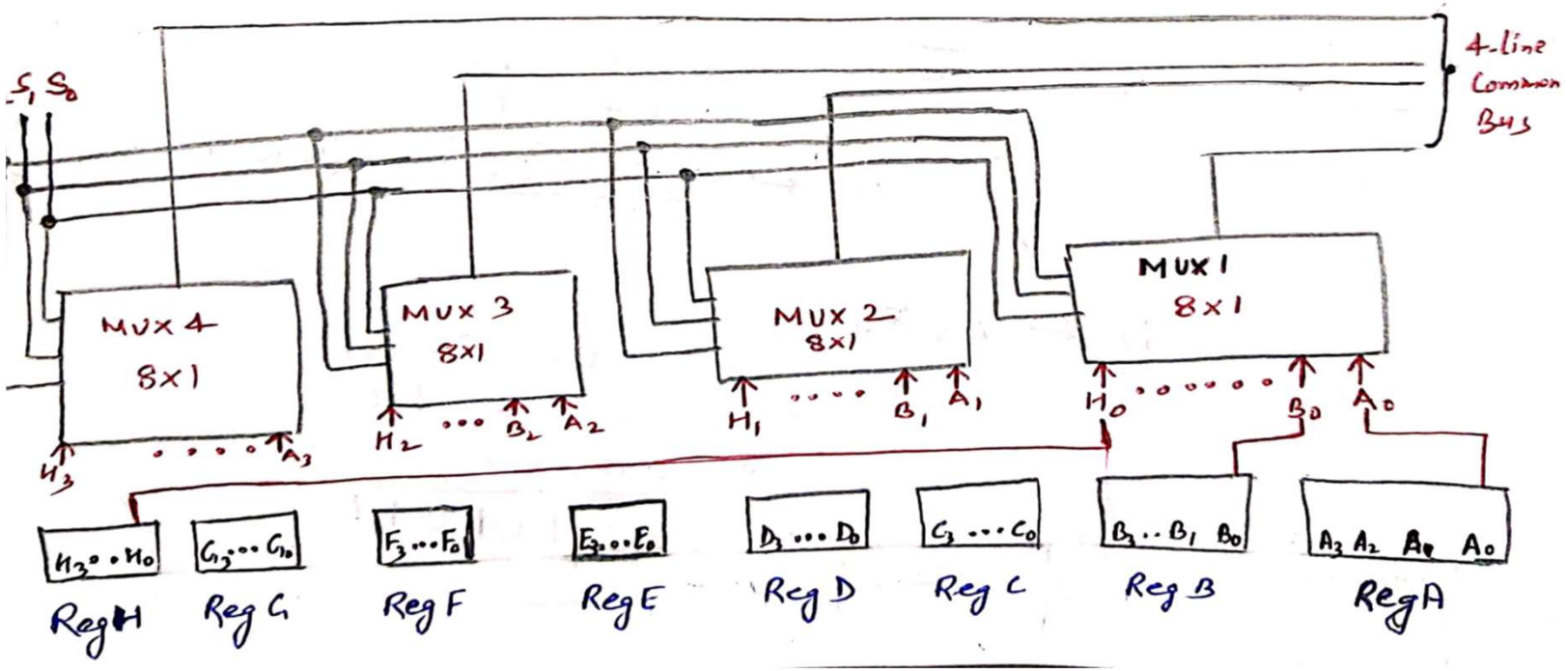
1. A bus organization has eight 4-bit registers. Show, how multiplexers can be used to select registers in a common bus configuration?

Answer:

Here, No. of MUX = No. of bits in each register = 4

Size of MUX = No. of register used × 1 = 8 × 1

Select lines for each MUX = 3 ($\because 8 = 2^3$)



Memory Transfer:

- The transfer of information from a memory word to the outside environment is called a read operation.
- The transfer of new information to be stored into the memory is called a write operation.
- A memory word will be symbolized by the letter M.
- Consider a memory unit that receives the address from a register called address register (AR) and data transferred to another register called data register (DR).
- The read operation can be symbolized by:

Read : DR ← M[AR]

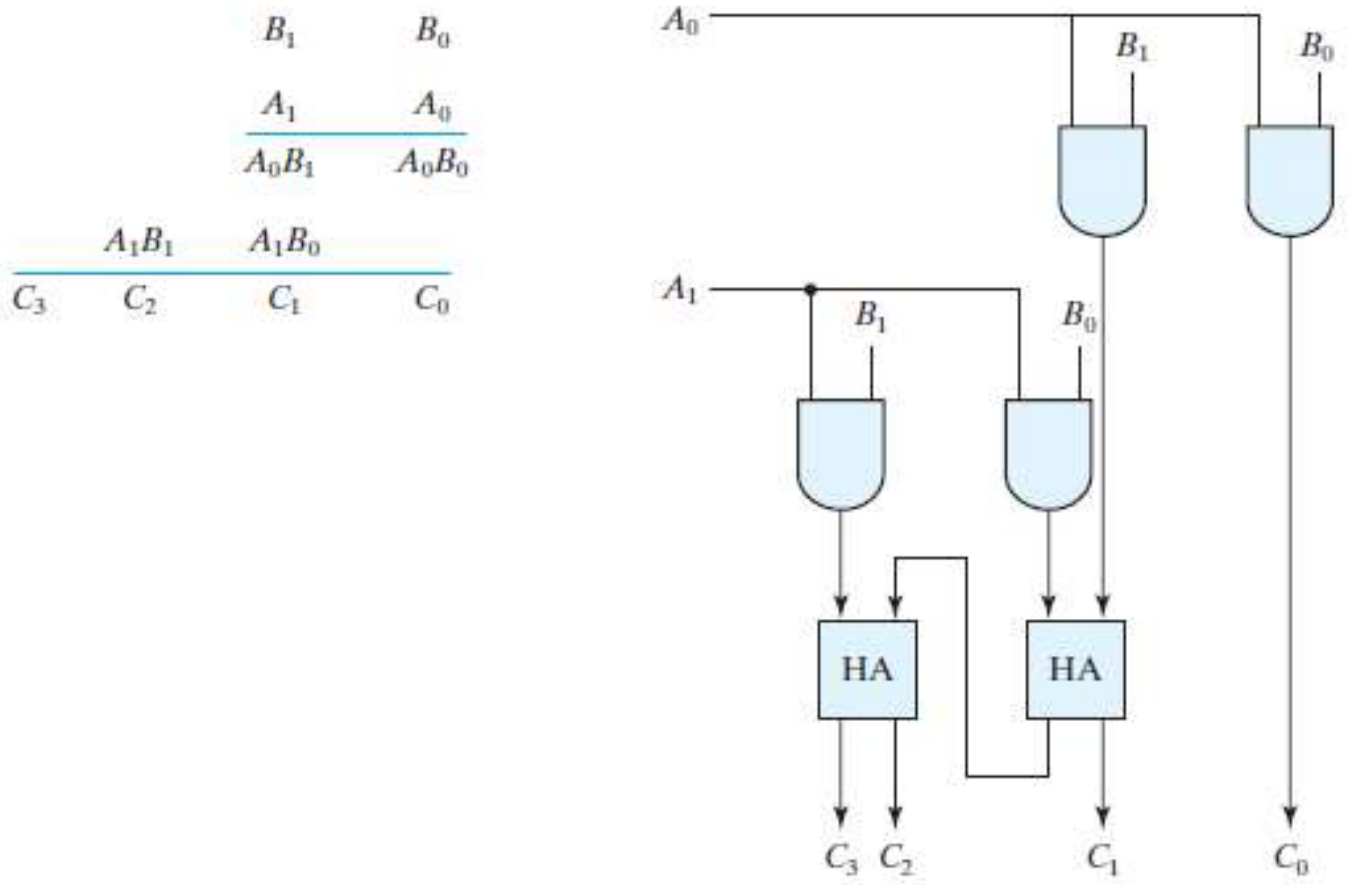
- This causes a transfer of information into DR from the memory word M selected by the address in AR.
- The write operation transfers the content of a data register like R1 to memory word M selected by address.

Write: M[AR] ← R1

- This causes a transfer of information from R1 into the memory word M selected by the address in AR.

Binary (Array) Multiplier:

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.



Two-bit by two-bit binary multiplier

Booth's Algorithm

- ✓ Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.

- ✓ As in all multiplication schemes, Booth algorithm requires examination of the multiplier bits and shifting of the partial product.

- ✓ Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to the following rules:

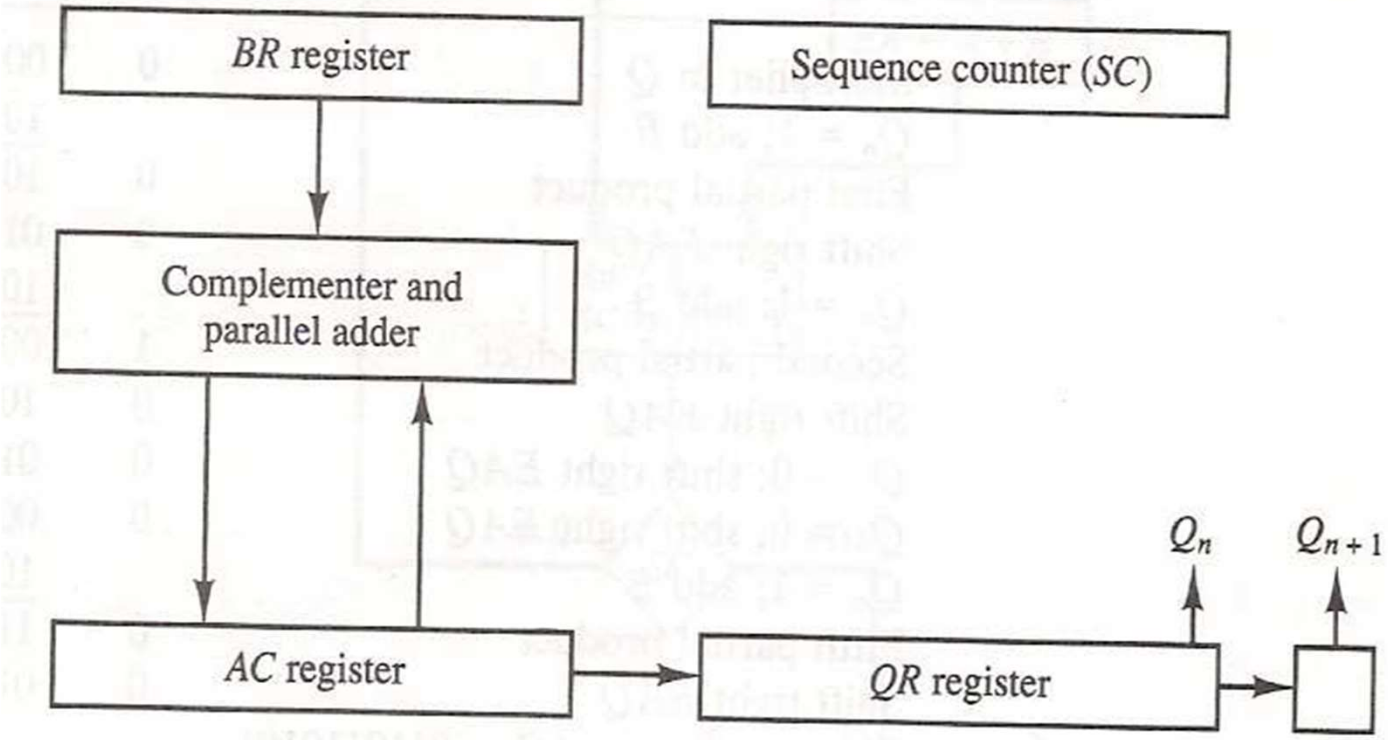
  1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
  2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of 0's in the multiplier.
  3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

- ✓ The algorithm works for positive or negative multipliers in 2's complement representation. This is because a negative multiplier ends with a string of 1's and the last operation will be a subtraction of the appropriate weight.

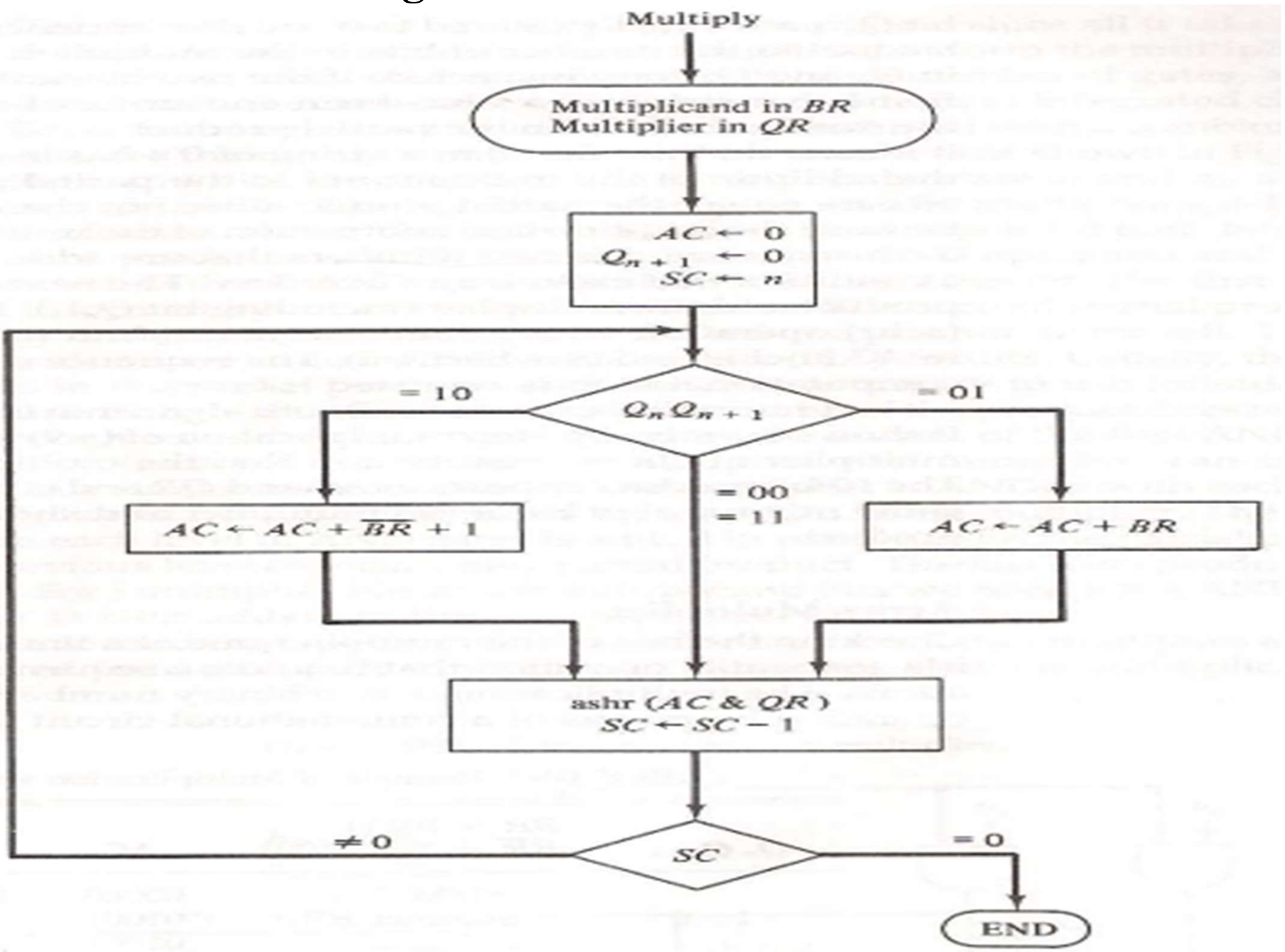
Hardware Implementation of Booth's Algorithm:

- The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.

- The multiplier and multiplicand are stored in the QR and BR registers respectively. Q_n designates the least significant bit of the multiplier in register QR. An extra flip-flop Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.



Flowchart of Booth's Algorithm:



Flowchart of Booth's Algorithm:

- AC and the appended bit Q_{n+1} are initially cleared to 0 and the sequence counter SC is set to a number n equal to the number of bits in the multiplier.

- The two bits of the multiplier in Q_n and Q_{n+1} are inspected.

- If the two bits are equal to 10, it means that the first 1 in a string of 1' s has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.

- If the two bits are equal to 01, it means that the first 0 in a string of 0' s has been encountered. This requires the addition of the multiplicand to the partial product in AC.

- When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the two numbers that are added always have opposite signs, a condition that excludes an overflow.

- The next step is to shift right the partial product and the multiplier (including bit Q_{n+1}). This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated n times.

Example of Booth's Algorithm:

- Example below shows the multiplication of **-9** and **-13** by Booth's Algorithm.

(-9) x (-13) = +117 (-9 = Multiplicand BR, -13 = Multiplier QR, Q_n= LSB of QR)

(-9 2's complement = 10111, -13 2's complement = 10011) (SC = n = 5 = 101)

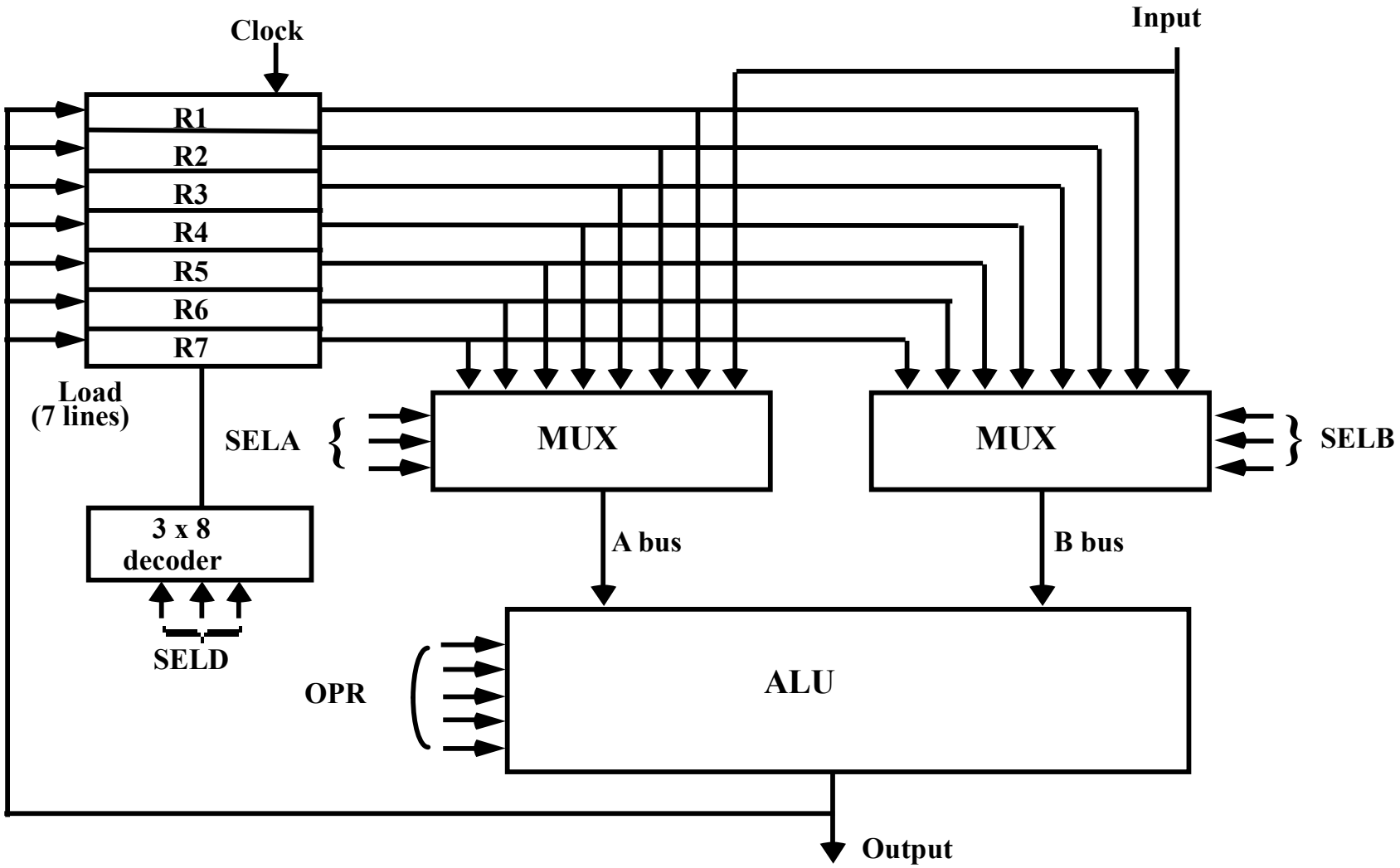
- The multiplier in QR is negative and that the multiplicand in BR is also negative. The 10-bit product appears in AC and QR and is positive. The final value of Q_{n+1} is the original sign bit of the multiplier and should not be taken as part of the product.

Q_n Q_{n+1}	BR = 10111 $\overline{BR}$ + 1 = 01001	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

- Final value = 00011 10101 = 1110101 = $(117)_{10}$

General Register Organization

- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. SELA, SELB, SELD OPR are control word.



Operation of Control Unit:

✓ The control unit directs the information flow through the registers and ALU by selecting various components in the system

Example: R1 ← R2 + R3

The control must provide binary selection variables to the following selector inputs:

[1] MUX A selector (SELA): BUS A ← R2
[2] MUX B selector (SELB): BUS B ← R3
[3] ALU operation selector (OPR): ALU to ADD
[4] Decoder destination selector (SELD): R1 ← Output Bus

Control Word (14 bit):

Encoding of register selection fields:

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

ALU Unit:

Encoding of ALU operations:

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of ALU Microoperations:

Microoperation	Symbolic Designation SELA	SELB	SELD	OPR	Control Word
R1 ← R2 - R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	-	R6	INCA	110 000 110 00001
R7 ← R1	R1	-	R7	TSFA	001 000 111 00000
Output ← R2	R2	-	None	TSFA	010 000 000 00000
Output ← Input	Input	-	None	TSFA	000 000 000 00000
R4 ← shl R4	R4	-	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

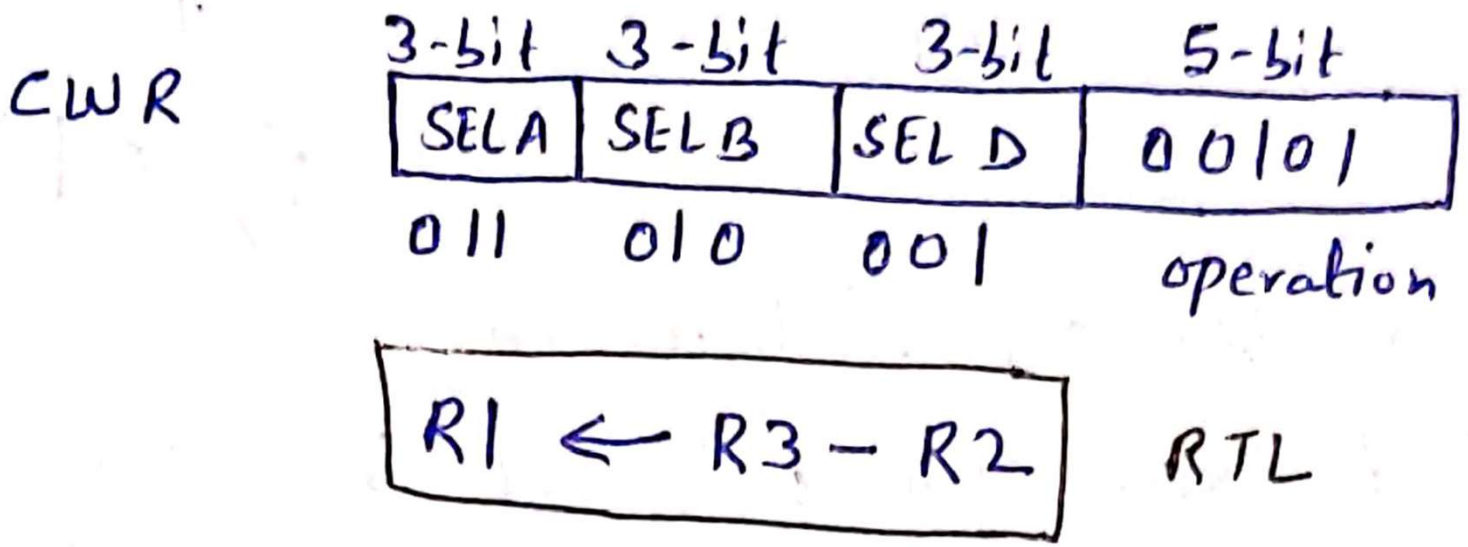
Problem:

1. **Considering the GRO configuration, write the RTL statement of the micro-operation which will be executed upon the control unit's generation of the control word 01101000100101.**

Answer:

Control word is given 01101000100101

RTL: SUB R1, R3, R2 (R1← R3-R2)



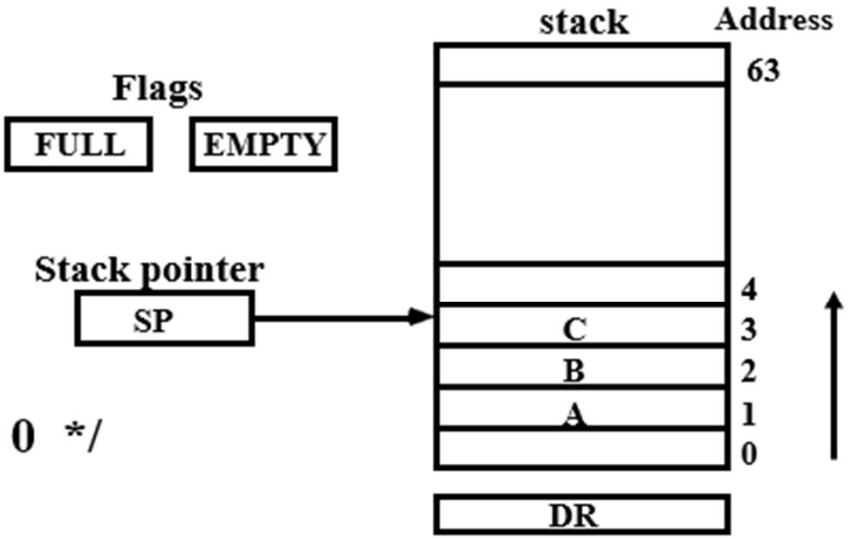
Stack Organization:

Stack:

- Efficient for arithmetic expression evaluation
- Storage which can be accessed in LIFO
- Pointer: SP (which holds the address for the stack is called stack pointer)
- Only PUSH (Insertion) and POP (Deletion) operations are applicable

Ex: 64-word stack, the SP contains 6 bits.

Register Stack



Push, Pop operations

/* Initially, SP = 0, EMPTY = 1, FULL = 0 */

PUSH	**POP**
SP ← SP + 1	**DR ← M[SP]**
M[SP] ← DR	**SP ← SP - 1**
If (SP = 0) then (FULL ← 1)	**If (SP = 0) then (EMPTY ← 1)**
EMPTY ← 0	**FULL ← 0**

Memory Stack:

- ✓ 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.

- ✓ Push operation:

$$SP \leftarrow SP - 1$$
$$M[SP] \leftarrow DR$$

- ✓ POP operation:

$$DR \leftarrow M[SP]$$
$$SP \leftarrow SP + 1$$

THANK YOU