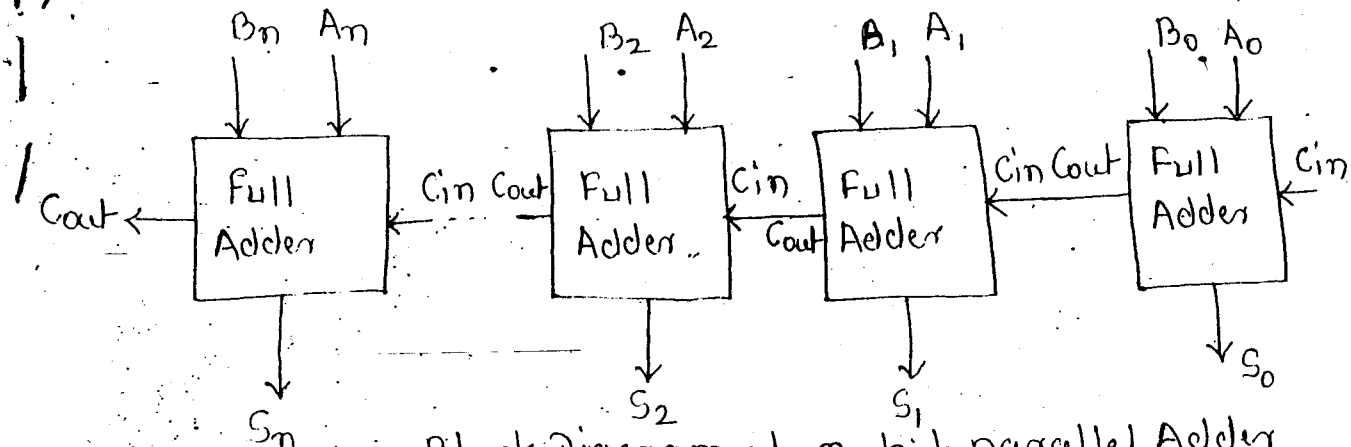
Parallel Adder :-



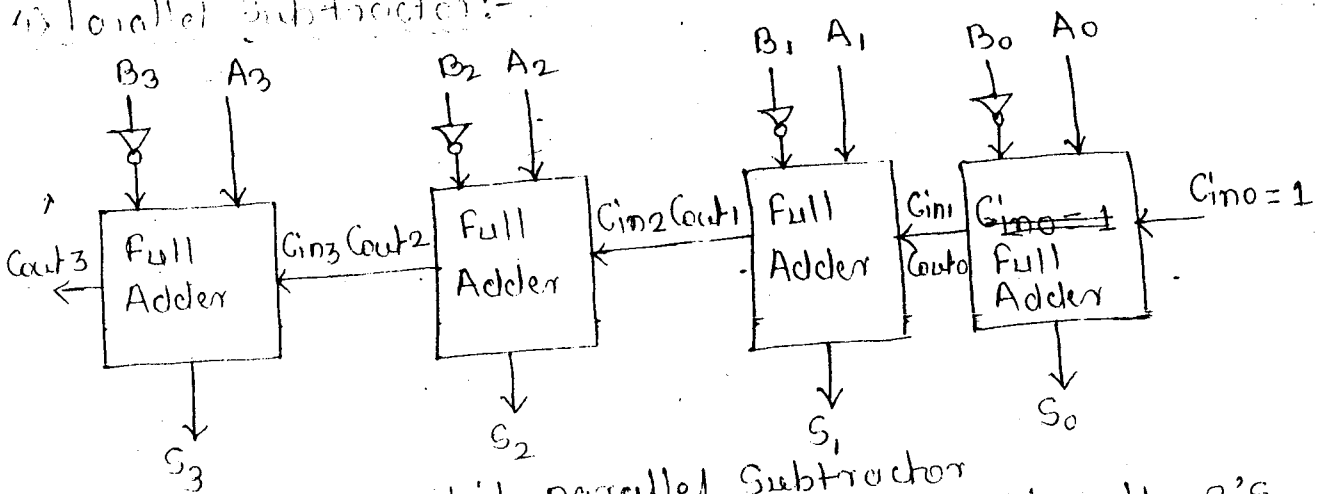
* Block Diagram of n-bit parallel Adder.

The Full adder consists of three inputs (A, B, Cin) & two outputs (Cout, Sum).

A n-bit parallel adder can be using no. of full adder circuits connected in parallel i.e the carry output of each adder is connected to the carry input of the next higher-order adder.

→ Carry input of a full-adder is made 0 because there is no carry into the least significant bit position.

Parallel Subtractor :-



* 4-bit parallel Subtractor

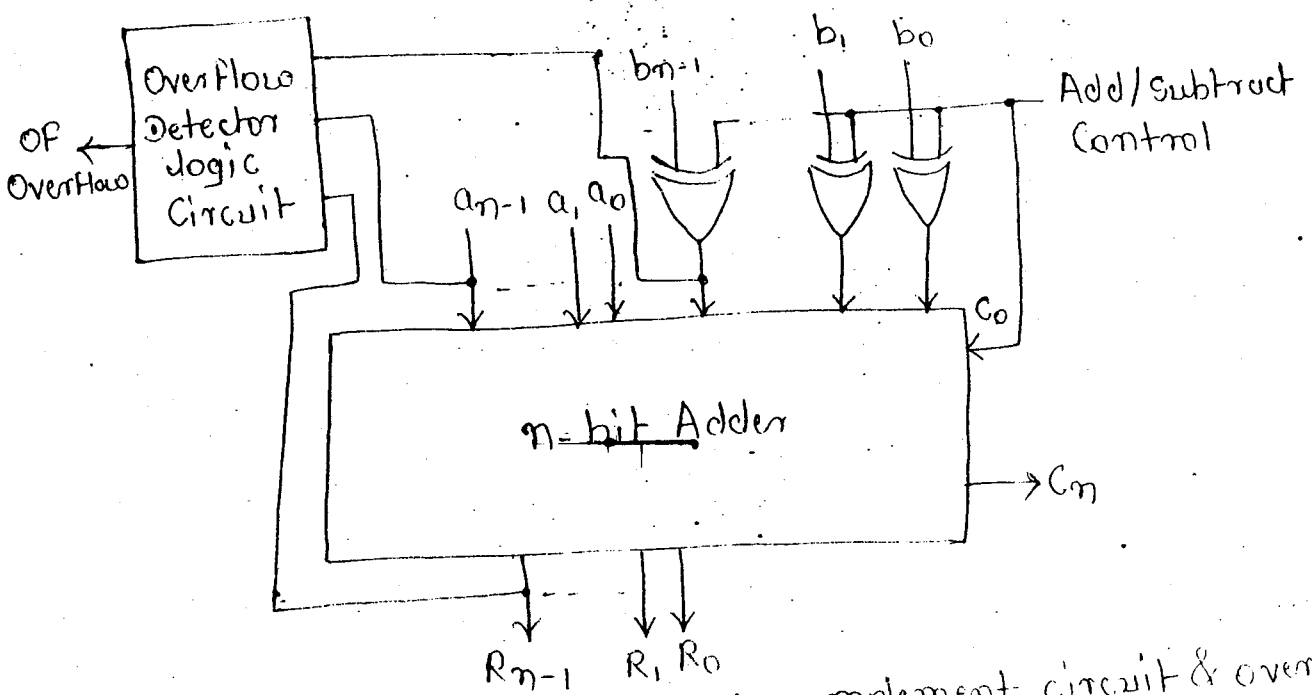
→ The subtraction A-B can be done by taking the 2's complement of B & adding it to A.

→ The 2's complement can be obtained by taking the 1's complement & adding one to the least significant pair of bits.

→ The 1's complemented can be implemented with inverters & a one can be added to the sum through the input carry.

5) Parallel Adder / Subtractor with overflow

* Hardware for integer addition & Subtraction:



→ It consists of n-bit adder, 2's complement circuit & overflow detector logic circuit.

→ Number a & Number b are the two inputs for n-bit adder.

→ For subtraction the number b is converted into its 2's complement form by making Add/Subtract control signal to the logic 1.

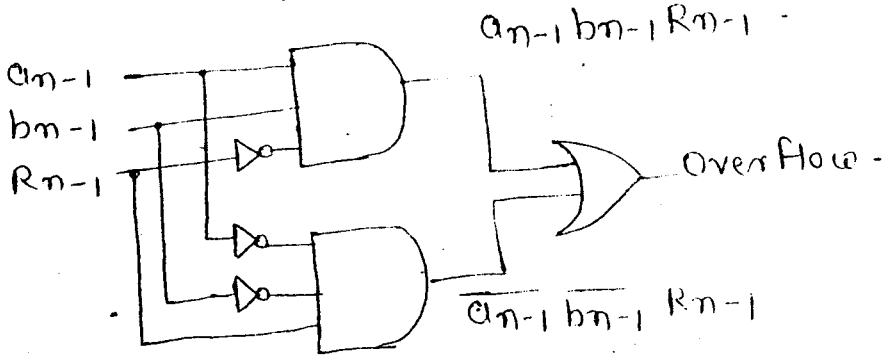
→ When Add/Subtract control signal is one all bits of number b are complemented & carry zero is set to 1.

∴ n-bit adder gives result as $R = a + \bar{b} + 1$

where $\bar{b}+1$ represents 2's complement of number b.

* Overflow detector logic circuit:

$$\text{Overflow} = a_{n-1} b_{n-1} \overline{R_{n-1}} + \overline{a_{n-1}}\ \overline{b_{n-1}}\ R_{n-1}$$



Carry Propagation delay: The sum & carry outputs of any stage cannot be produced until the input carry occurs, this leads to a time delay in the addition process This delay is known as Carry Propagation delay.

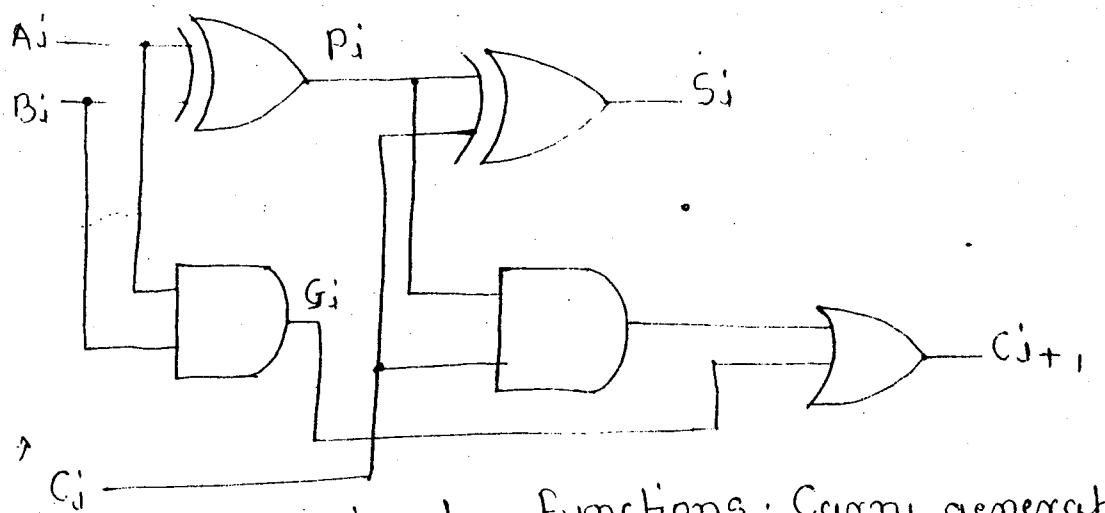
For ex:-

$$\begin{array}{r} {\scriptstyle 1\,1\,1\,} \\ 0101 \\ +\,0011 \\ \hline 1000 \end{array}$$

Speeding up this process by eliminating inter stage carry delay is called Carry Look ~~Ahead~~ Ahead Adder.

It uses two functions: Carry generate & Carry propagate

→ Full-adder Circuit:



We define two functions: Carry generate (G_i) & carry ~~propagate~~ (P_i)

$$P_i = A_i \oplus B_i = A_i\bar{B_i} + \bar{A_i}B_i$$

$$G_i = A_i B_i$$

The output Sum (S_i):

$$S_i = P_i \oplus C_i = P_i\bar{C_i} + C_i\bar{P_i}$$

$$C_{i+1} = G_i + P_i C_i$$

$$= A_i B_i + (A_i \oplus B_i) C_i$$

$$= A_i B_i + (\bar{A_i} B_i + A_i \bar{B_i}) C_i$$

$$= A_i B_i + \bar{A_i} B_i C_i + A_i \bar{B_i} C_i$$

$$= A_i B_i (C_{i+1}) + \bar{A_i} B_i C_i + A_i \bar{B_i} C_i \quad (\because C_i + 1 = 1)$$

$= A_i B_i C_i + A_i B_i + \bar{A_i} B_i C_i + A_i \bar{B_i} C_i$

$= \cancel{A_i B_i (C_{i+1})}$

$= A_i B_i + A_i C_i (B_i + \bar{B_i}) + \bar{A_i} B_i C_i$

$= A_i B_i + A_i C_i + \bar{A_i} B_i C_i$

$= A_i B_i + A_i C_i + \bar{A_i} B_i C_i$

$= A_i B_i (C_i + 1) + A_i C_i + \bar{A_i} B_i C_i$

$= A_i B_i C_i + A_i B_i + A_i C_i + \bar{A_i} B_i C_i$

$= B_i C_i (A_i + \bar{A_i}) + A_i B_i + A_i C_i$

$\boxed{C_{i+1} = A_i B_i + A_i C_i + B_i C_i}$ $\quad C_{i+1} = A_i B_i + (A_i + B_i) C_i$

$= A_i B_i + P_i C_i$

$\therefore P_i = (A_i + B_i)$

$\therefore \boxed{C_{i+1} = G_i + P_i C_i}$ — Carry

$S_i = P_i \oplus C_i$

$= P_i \oplus C_i = (A_i \oplus B_i) \oplus C_i$

$= [A_i \bar{B_i} + B_i \bar{A_i}] \oplus C_i$

$= P_i \cdot \bar{C_i} + \bar{P_i} C_i$

$= (A_i \oplus B_i) \bar{C_i} + (\overline{A_i \oplus B_i}) C_i$

$= (A_i \cdot \bar{B_i} + \bar{A_i} B_i) \bar{C_i} + (A_i \odot B_i) C_i$

$= A_i \bar{B_i} \bar{C_i} + \bar{A_i} B_i \bar{C_i} + (A_i B_i + \bar{A_i} \bar{B_i}) C_i$

$\boxed{S_i = A_i \bar{B_i} \bar{C_i} + \bar{A_i} B_i \bar{C_i} + A_i B_i C_i + \overline{A_i B_i} C_i}$

$\boxed{S_i = P_i \oplus C_i}$

G_i: Carry generate & it produces on carry when both A_i & B_i are one, regardless of the input carry.

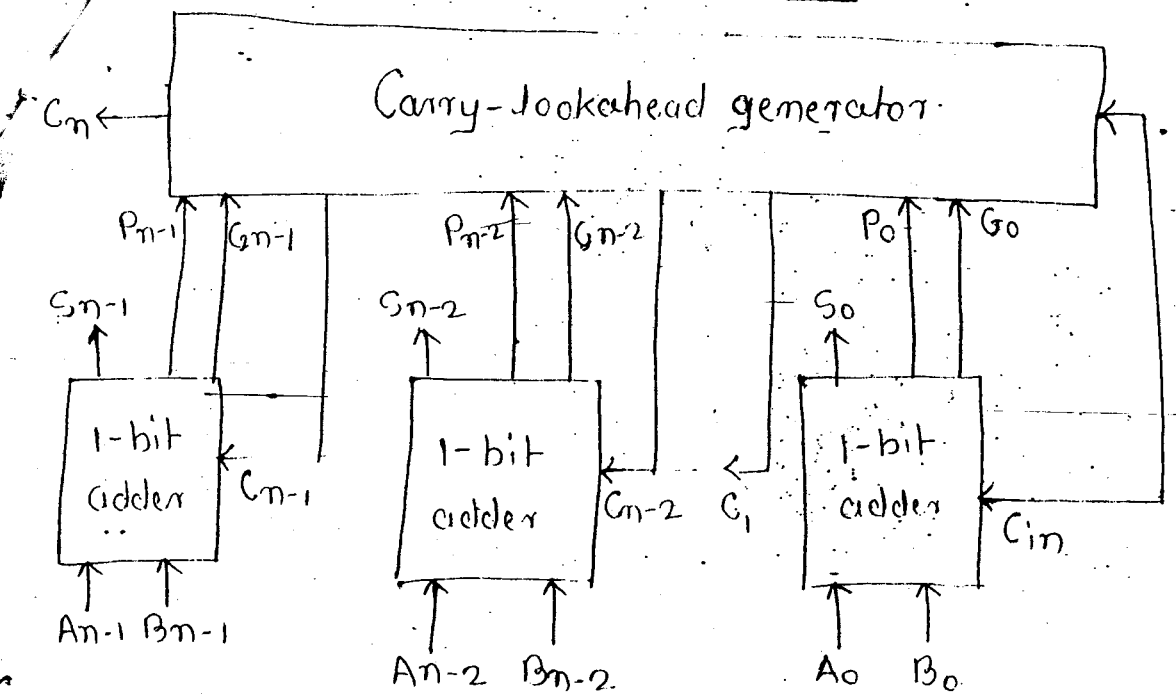
P_i: Carry propagate because it is term associated with the Propagation of the carry from C_i to C_{i+1}

C_{i+1}: Sum of products function of the P & G outputs of all the preceding stages.

Carry - look ahead adder can be expressed as:

$C_1 = G_0 + P_0 C_{in}$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$



* Carry lookahead adder circuit

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{in}$$

$$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{in}.$$

Let us assume,

$$S_i = P_i \oplus G_i \oplus C_i$$

$$P_i = A_i + B_i$$

$$G_i = A_iB_i$$

$$S_i = (A_i + B_i) \oplus A_iB_i \oplus C_i$$

$$= \left[\overline{(A_i + B_i)}\,(A_iB_i) + (A_i + B_i)\,\overline{(A_iB_i)}\right] \oplus C_i$$

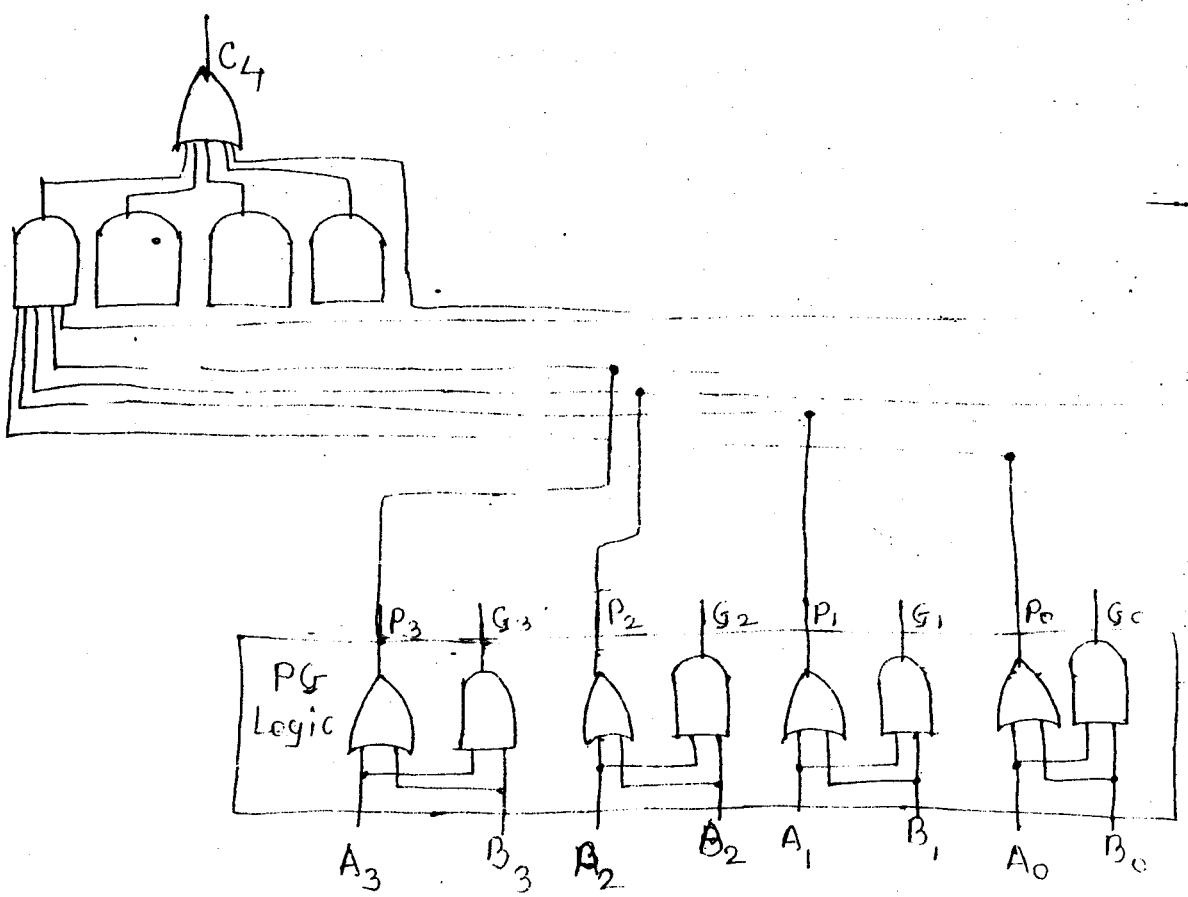
$$= \left[\overline{A_i}\,\overline{B_i}\,(A_iB_i) + (A_i + B_i)\,(\overline{A_i} + \overline{B_i})\right] \oplus C_i$$

$$= \left[0 + A_i\overline{A_i} + A_i\overline{B_i} + B_i\overline{A_i} + B_i\overline{B_i}\right] \oplus C_i$$

$$= (A_i\overline{B_i} + B_i\overline{A_i}) \oplus C_i$$

$$\boxed{S_i = A_i \oplus B_i \oplus C_i}$$

Carry-lookahead adder Circuit:
4-bit)



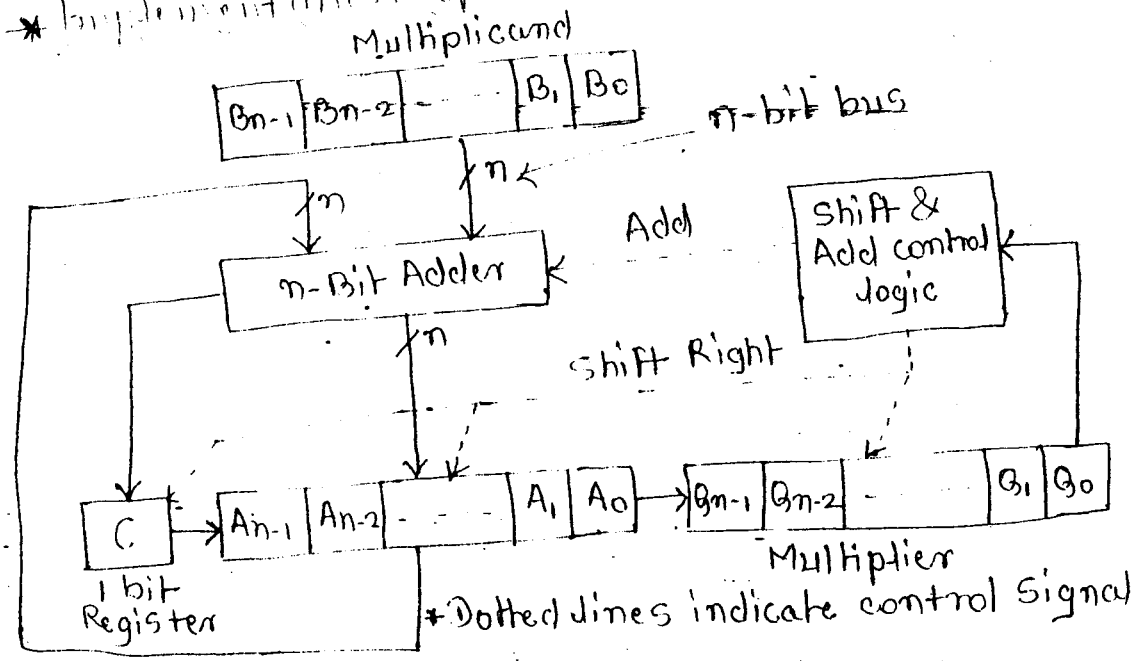
* Manual multiplication Algorithm:

```
      1101      (13) Multiplicand
      1001      (9) Multiplier
    ------
      1101
     0000
    0000
   1101        } Partial Products
  ---------
  1110101      Final Product (117)
```

* Steps:-

1) Multiplication process involves generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.
2) In the binary system the partial products can be defined as when the multiplier bit is 0, the partial product is 0, & when the multiplier bit is 1, the partial product is the multiplicand.
3) The final product is produced by summing the partial products. Before summing operation each successive partial product is shifted one position to the left relative to the preceding partial product.
4) The product of 2n-digit no. can be 2n digits.

* Implementation of manual multiplication:-



Multiplication Operation Steps:

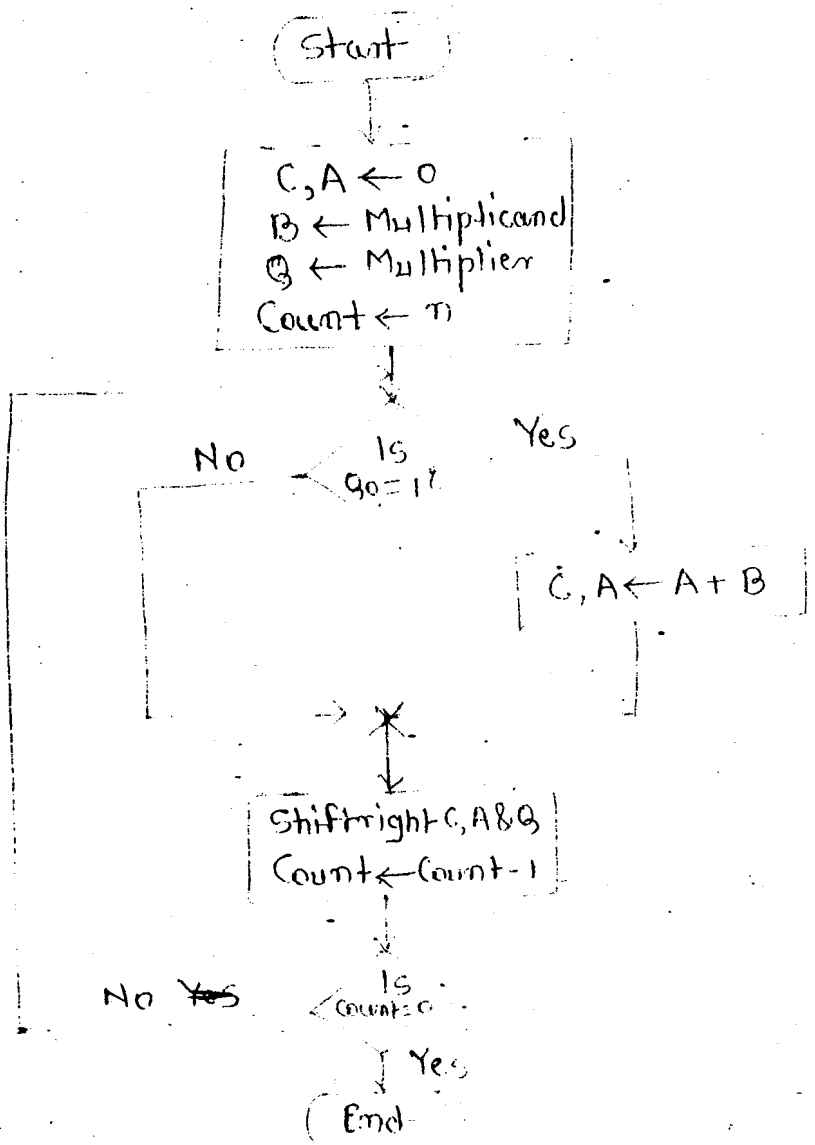
It consists of n-bit binary adder, shift & add control logic & four registers, A, B, C & Q. Multiplier & Multiplicand are loaded into register Q & register B. & C are initially set to 0.

1) If bit 0 (Q_0) is one then multiplicand & partial product are added & all bits of C, A & Q registers are shifted to the right one bit, so that the C bit goes into A_{n-1}, A_0 goes into Q_{n-1}, & Q_0 is lost.

2) If bit 0 (Q_0) is 0, then no addition is performed, only shift operation is carried out.

∴ Similarly steps 1 & 2 repeated n-times.

* Flowchart for multiplication operation:-

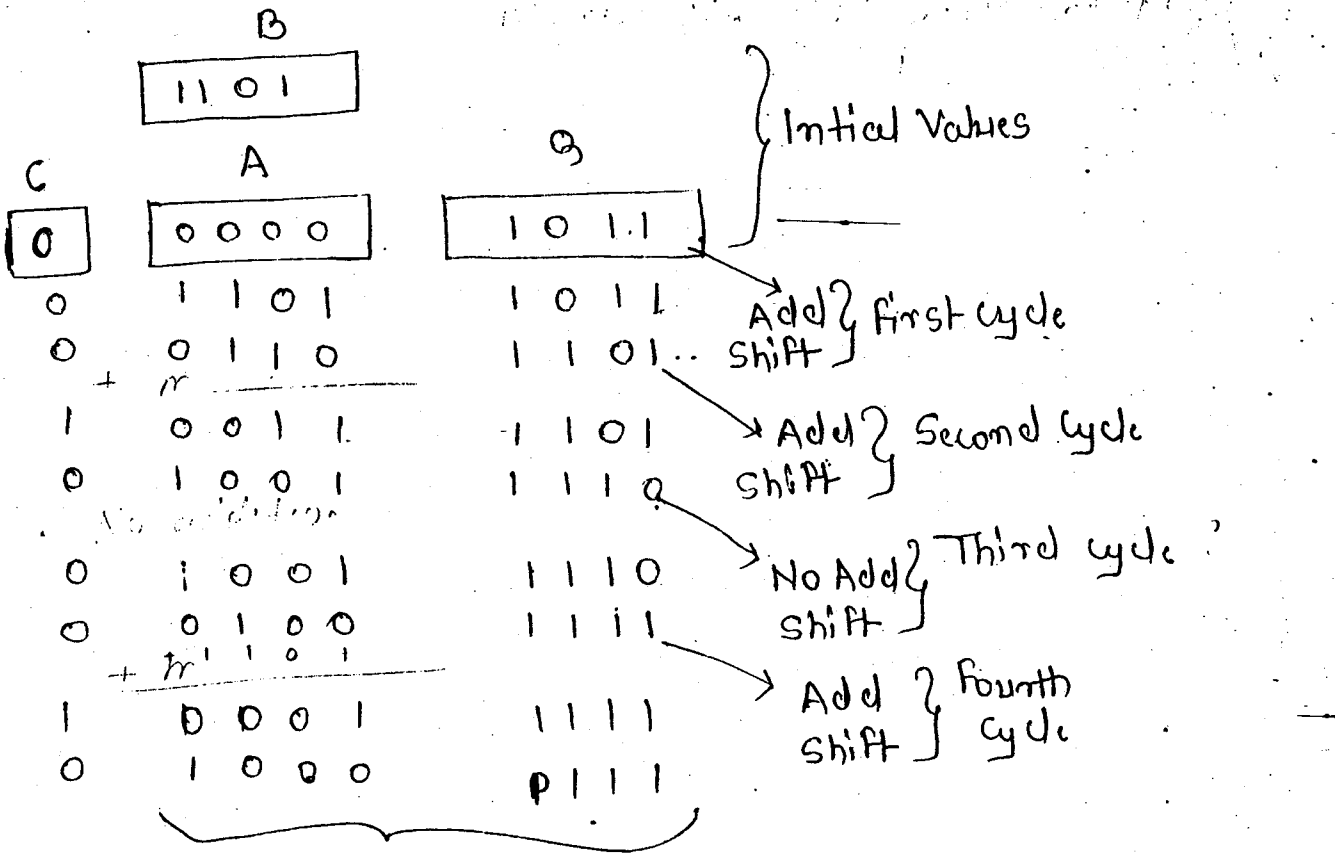


For example:-

consider 4-bit multiplier & multiplicand:

(A) Multiplier = 1101 & (Q) Multiplier = 1011

(B) Multiplicand = 1011 & (B) Multiplicand = 1101



- Booth's Algorithm:-

A powerful algorithm for signed-number multiplication is a Booth's algorithm, which generates a 2n-bit product & treats both positive & negative numbers uniformly.

Booth ~~multiplier~~ recording table:

Multiplier		Version of Multiplicand
Bit i	Bit i-1	Selected by bit i
0	0	0 × M
0	1	+1 × M
1	0	−1 × M
1	1	0 × M

(M (Multiplicand))

for ex:- Recode the multiplier 101100 for booth's multiplication:

1 0 1 1 0 0

∴ Assume 0 to the right of the multiplier LSB.

1 0 1 1 0 0 [0] Multiplier

-1 +1 0 -1 0 0 Recoded Multiplier.

2) 0 1 1 0 0 1

0 1 1 0 0 1 [0]

+1 0 -1 0 +1 -1 Recoded Multiplier

* Booth's multiplication:-

Q = Multiplier : 0 0 1 1 0 0 Multiplicand (B): 0 1 0 0 1 1

Recoded multiplier: 0 +1 0 -1 0 0

Multiplication:

0 1 0 0 1 1

0 +1 0 -1 0 0

0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0		
1	1	1	1	1	0	1	0	← 2's complement of the multiplicand.			
0	0	0	0	0	0	0	0				
0	0	0	1	0	0	1	1				
0	0	0	0	0	0	0					

0 0 0 0 1 1 1 0 0 1 0 0

* Whenever multiplicand is multiplied by -1 its 2's complement is taken as a partial result.

0 10011 → 101100 (+1) → 101101

(positive to positive multiplier).

* The same algorithm can be used for negative multiplier.

* Multiply 01110 (+14) & 11011 (-5)

Soln:- 0 1 1 1 0 (+14) Multiplicand

1 1 0 1 1 (-5) Multiplier

0 -1 +1 0 -1 Recoded Multiplier

Multiplication:

0 1 1 1 0

0 -1 +1 0 -1

1	1	1	1	1	1	0	0	1	0
0	0	0	0	0	0	0	0	0	
0	0	0	0	1	1	1	0		
1	1	1	0	0	0	1			
0	0	0	0	0	0				

1 1 1 0 1 1 1 0 1 0 (-70)

0 1 1 1 0

1 0 0 0 1

1

1 0 0 1 0

← 2's Complement of the multiplicand

same algorithm also can be used for negative multiplier & negative multiplicand.

Multiplicand: 1 1 0 0 1 1 ~~(-13)~~ (-19)

Multiplier : 1 0 1 1 0 0 (-12)

Recoded Multiplier: -1 +1 0 -1 0 0

Multiplication:

1 1 0 0 1 1

-1 +1 0 -1 0 0

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	
1	1	1	1	0	0	1	1	0	1		
0	0	0	0	0	0	0	0	0			
0	0	1	1	0	0	1	1				
1	0	0	1	1	0	1					
1	0	1	1	1	1	0	0	0	1	0	0

1 1 0 0 1 1

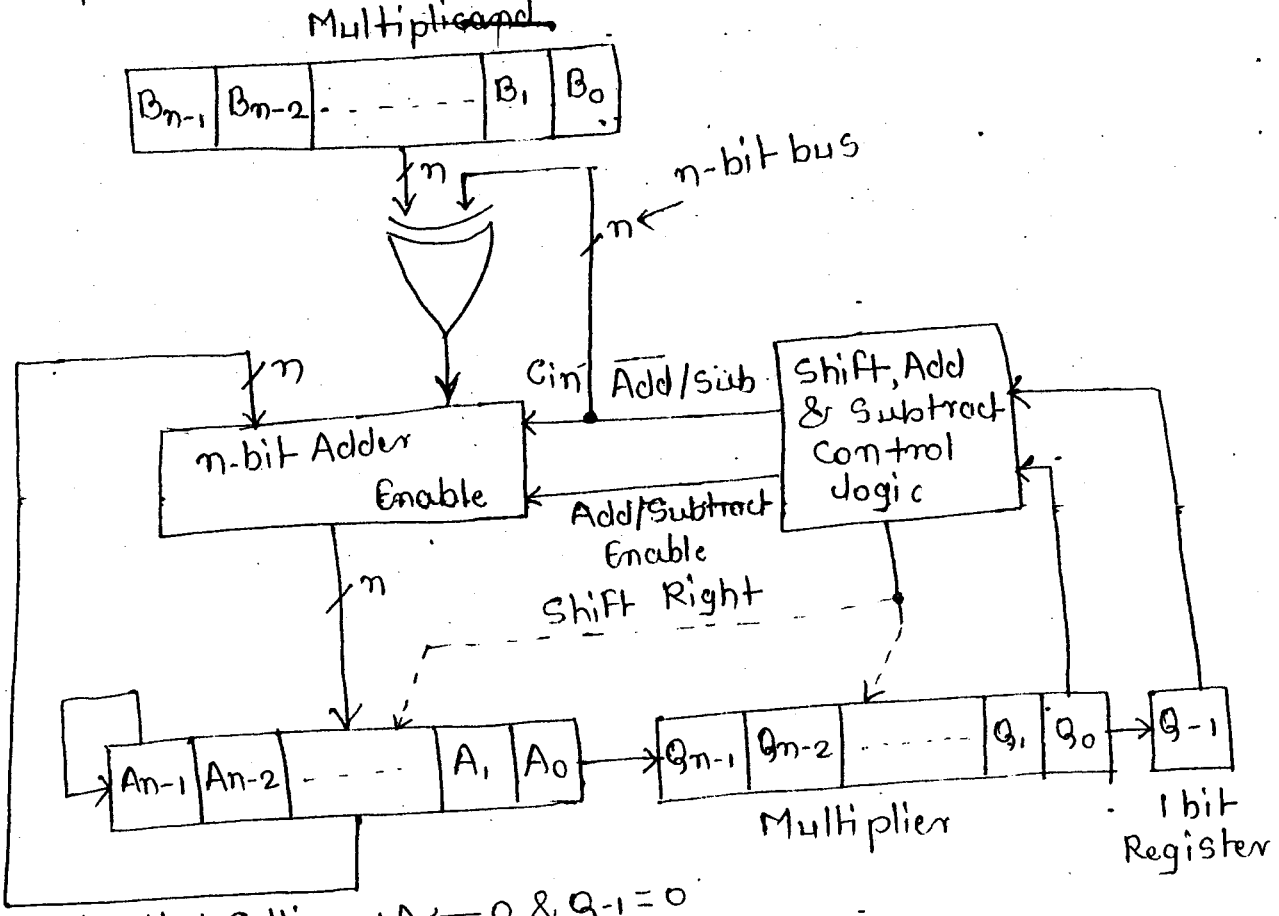
0 0 1 1 0 0

1

0 0 1 1 0 1

← 2's complement of the multiplicand.

* Implementation of signed binary multiplication:-



* Initial Settings: A ← 0 & Q-1 = 0

+ It consists of n-bit adder, shift, add Subtract control logic & four registers, A, B, Q & Q_{-1}. Multiplier & multiplicand are loaded into register Q & register B. & register A & Q_{-1} are initially set to 0. Control logic scans the bits of the multiplier &

→ The n-bit adder performs addition of two inputs. One input is the A & other input is multiplicand

→ In case of addition, $\overline{Add}$/Sub line is 0 ∴ Cin = 0 & multiplicand is directly applied as a second input to the n-bit adder.

→ In case of Subtraction, $\overline{Add}$/Sub line is 1 ∴ Cin = 1 & multiplicand is complemented & then applied to the n-bit adder. As a result, the 2's complement of multiplicand is added in the A register.

* Truth table for Shift, add & Subtract control logic:

Q_0	Q_{-1}	$\overline{Add}$/Sub	Add/Subtract Enable	Shift
0	0	x	0	1
0	1	0	1	1
1	0	1	1	1
1	1	x	0	1

1) If two bits are same, then all the bits of the A, Q & Q_{-1} registers are shifted to right 1 bit with out addition or Subtraction (Add/subtract Enable = 0). ($Q_0 = 0, 1$ or 1)

2) If $Q_0 = 0$ & $Q_{-1} = 1$ then multiplicand is added & if bits are $Q_0 = 1$ & $Q_{-1} = 0$ then multiplicand is Subtracted. After addition

* Flowchart: -

Multiplication of 4-bit Numbers:

Case 1) Both Positive (5×4)

Multiplicand(B) ← 0101 (5) Multiplier (Q) ← 0100 (4)

Steps	A	Q	Q-1	Operation
	0000	0100	0	Intial
Step 1:	0000	0010	0	Shift right
Step 2:	0000	0001	0	Shift right
Step 3:	1011	0001	0	A ← A-B
	1101	1000	1	Shift right
Step 4:	0010	1000	1	A ← A+B
	0001	0100	0	Shift right

0101 — 1's: 1010, +1 → 1011

(1101 + 0101) → 0010

Result: 0001 0100 = +20

Case 2) Negative Multiplier (5× -4)

Multiplicand (B) ← 0101 (5) Multiplier (Q) ← 1100 (-4)

Steps	A	Q	Q-1	Operation
	0000	1100	0	Intial
Step 1:	0000	0110	0	Shift right
Step 2:	0000	0011	0	Shift right
Step 3:	1011	0011	0	A ← A-B
	1101	1001	1	Shift right
Step 4:	1110	1100	1	Shift right

0101 → 1010, +1 → 1011

Result: 1110 1100 = (-20) (2's complement of +20)

(0001 0100 → 1110 1011 → 1111 → 1110 1100)

Case 3) Negative Multiplicand (-5×4) -6×3

Multiplicand (B) ← 1011 Multiplier (Q) ← 0100 (4)

Steps	A	Q	Q-1	Operation
	0000	0100	0	Intial
Step 1:	0000	0010	0	Shift right
Step 2:	0000	0001	0	Shift right
Step 3:	0101	0001	0	A ← A-B
	0010	1000	1	Shift right
Step 4:	1101	1000	1	A ← A+B
	1110	1100	0	Shift right

1011 → 0100 → 1 → 0101

0010 + 1011 → 1101

Result: 1110 1100 = (-20) (2's complement of +20)

Case 4): (-5×4) Both Negative:

Multiplicand (B) ← 1011 (-5) Multiplier (Q) ← 1100 (-4)

Steps	A	Q	Q-1	Operation
	0000	1100	0	Intial
Step 1:	0000	0110	0	Shift right
Step 2:	0000	0011	0	Shift right
Step 3:	0101	0011	0	A ← A - B
	0010	1001	1	Shift right
step 4:	0001	0100	1	shift right

Result: 0001 0100 = +20

1011
0100
0101

* Modified Booth's Algorithm:-

To speed-up the multiplication process in the Booth's algorithm a technique called bit-pair recoding is used. It is also called Modified Booth's algorithm.

→ Table for the bit-pair code for all possible multiplier bit operations:

Multiplier bit-pair		Multiplier bit on the right	Bit-pair Recoded multiplier bit at position i
i+1	i	i-1	
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

* Find the bit pair code for multiplier:

11010

By referring table

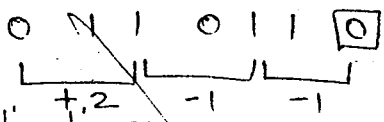
Sign extension → [1] 1 1 0 1 0 [0] Implied 0 to right of LSB

Multiply given signed 2's complement numbers using bit pair recoding.

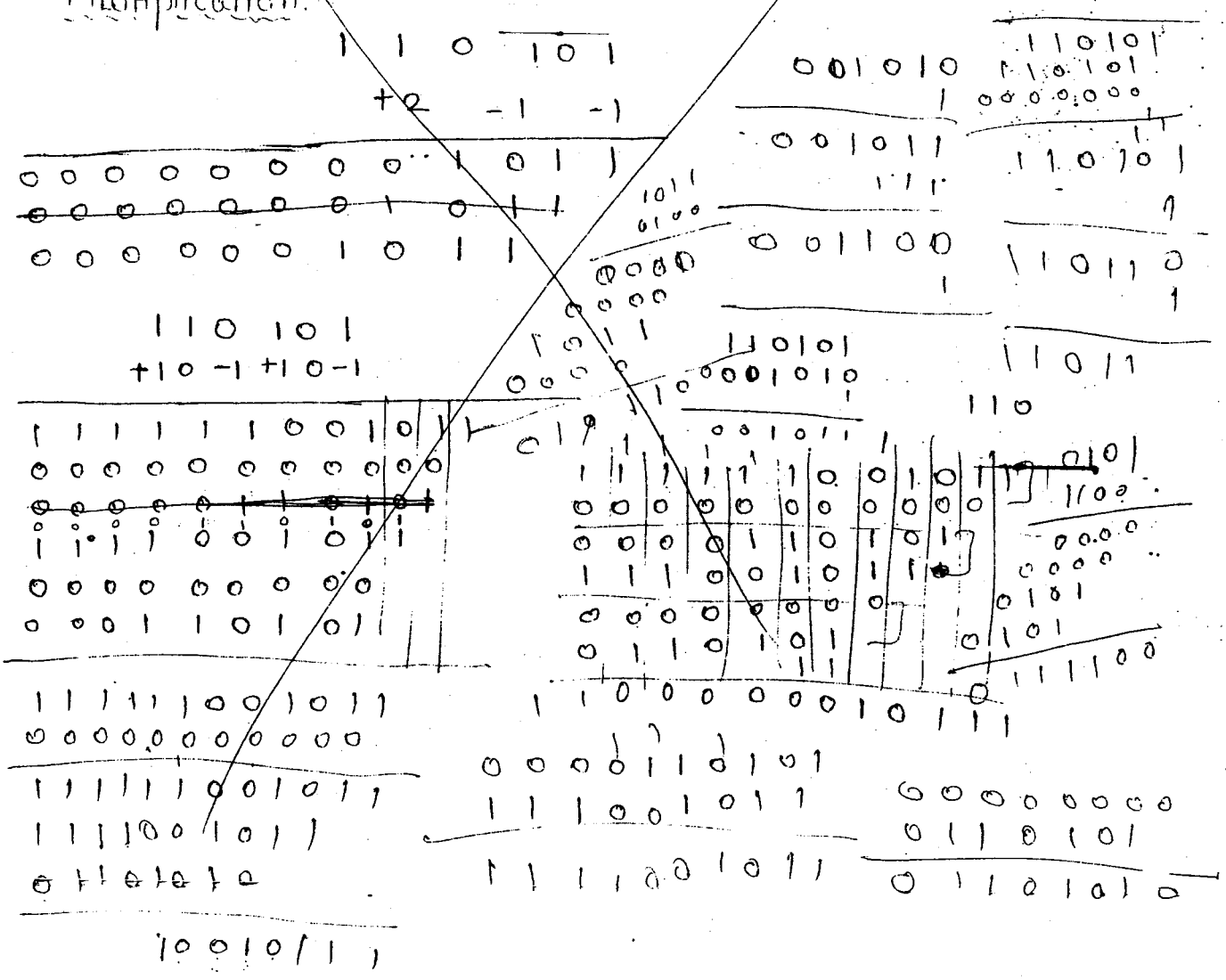
A = 110101 multiplicand (-11)

B = 011011 multiplier (+27) → +10-1+10-1

By referring table



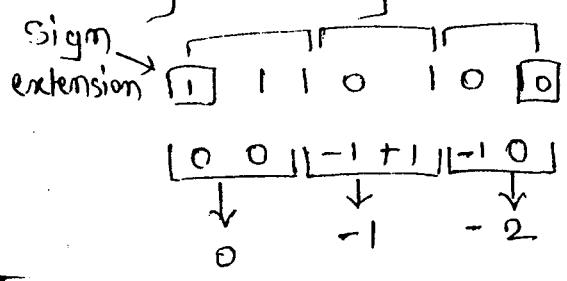
Multiplication:



A = 01101 (+13) = Multiplicand = B

B = 11010 (-6) = Multiplier = Q

By referring table

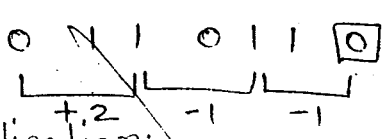


Multiply given signed 2's complement numbers using bit pair recoding.

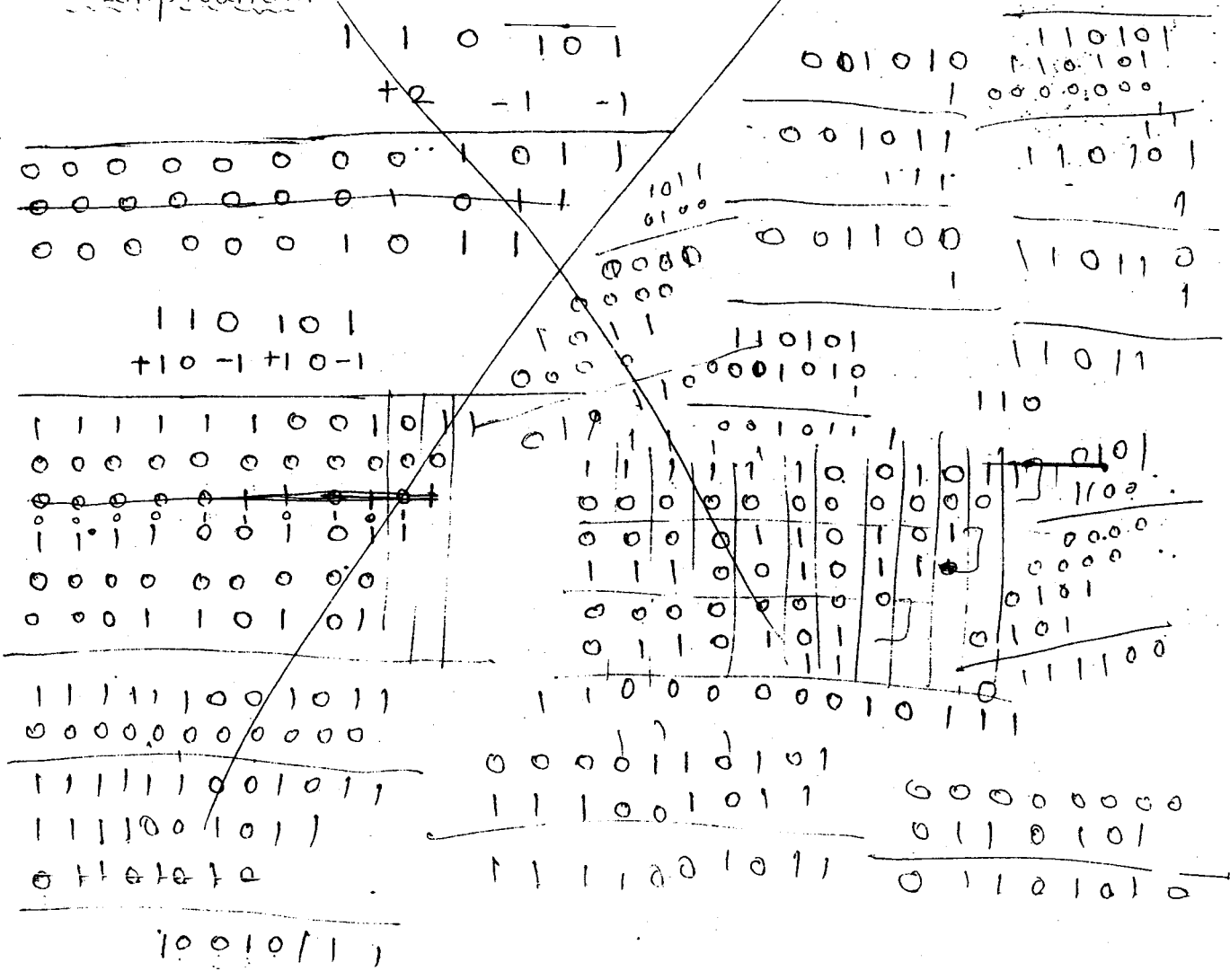
A = 110101 multiplicand (-11)

B = 011011 multiplier (+27) → +1 0 -1 +1 0 -1

By referring table



Multiplication:



---

* A = 01101 (+13) = Multiplicand = B

B = 11010 (-6) = Multiplier = Q

By referring table