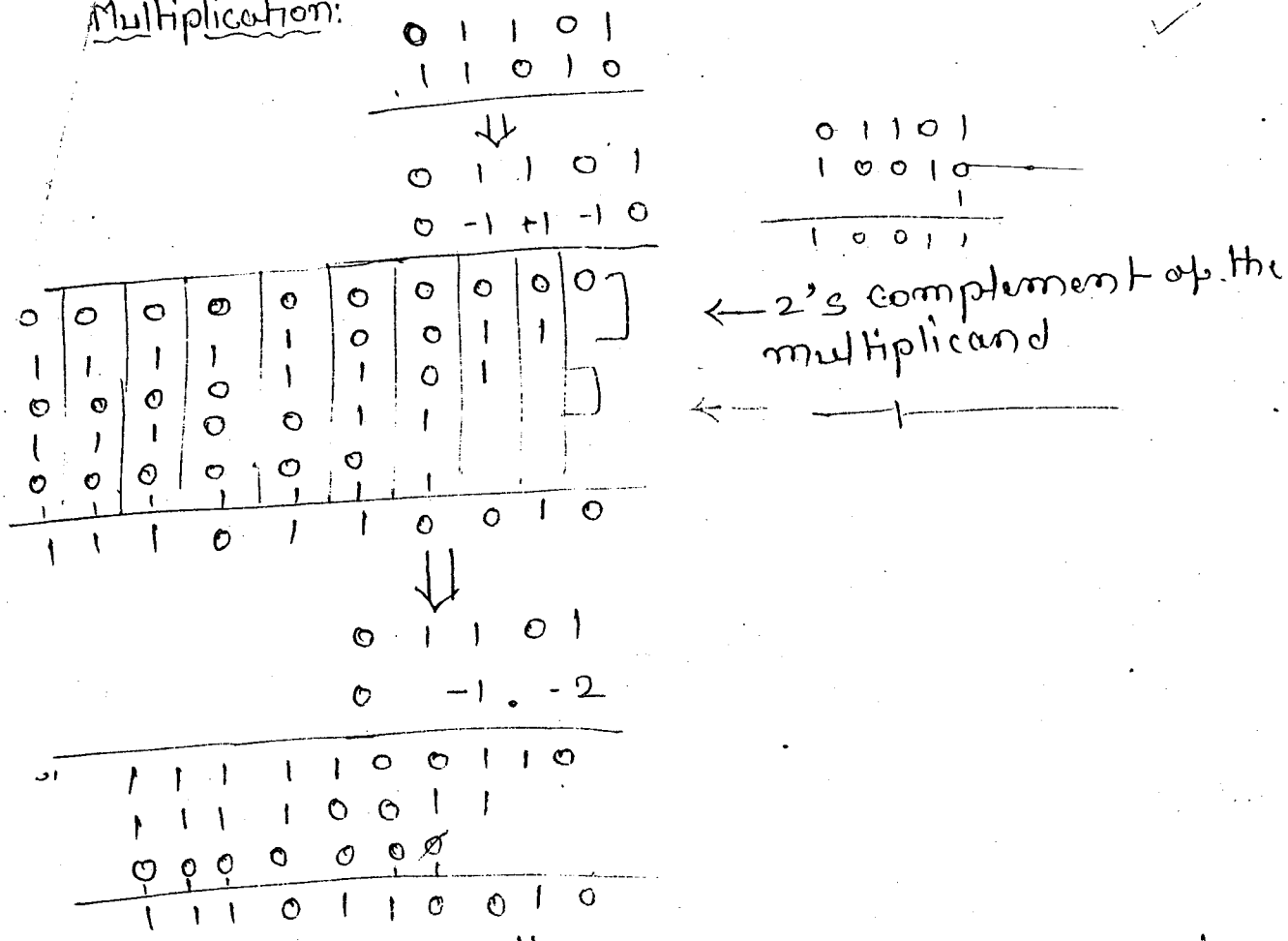
Multiplication:



★ Combinational Multipliers:-

It uses n shifts & add to multiply n-bit binary number. The combination logic ckt implemented to perform such multiplication is called combinational multiplier.

★ Multiplication process of 4×4 multiplier:

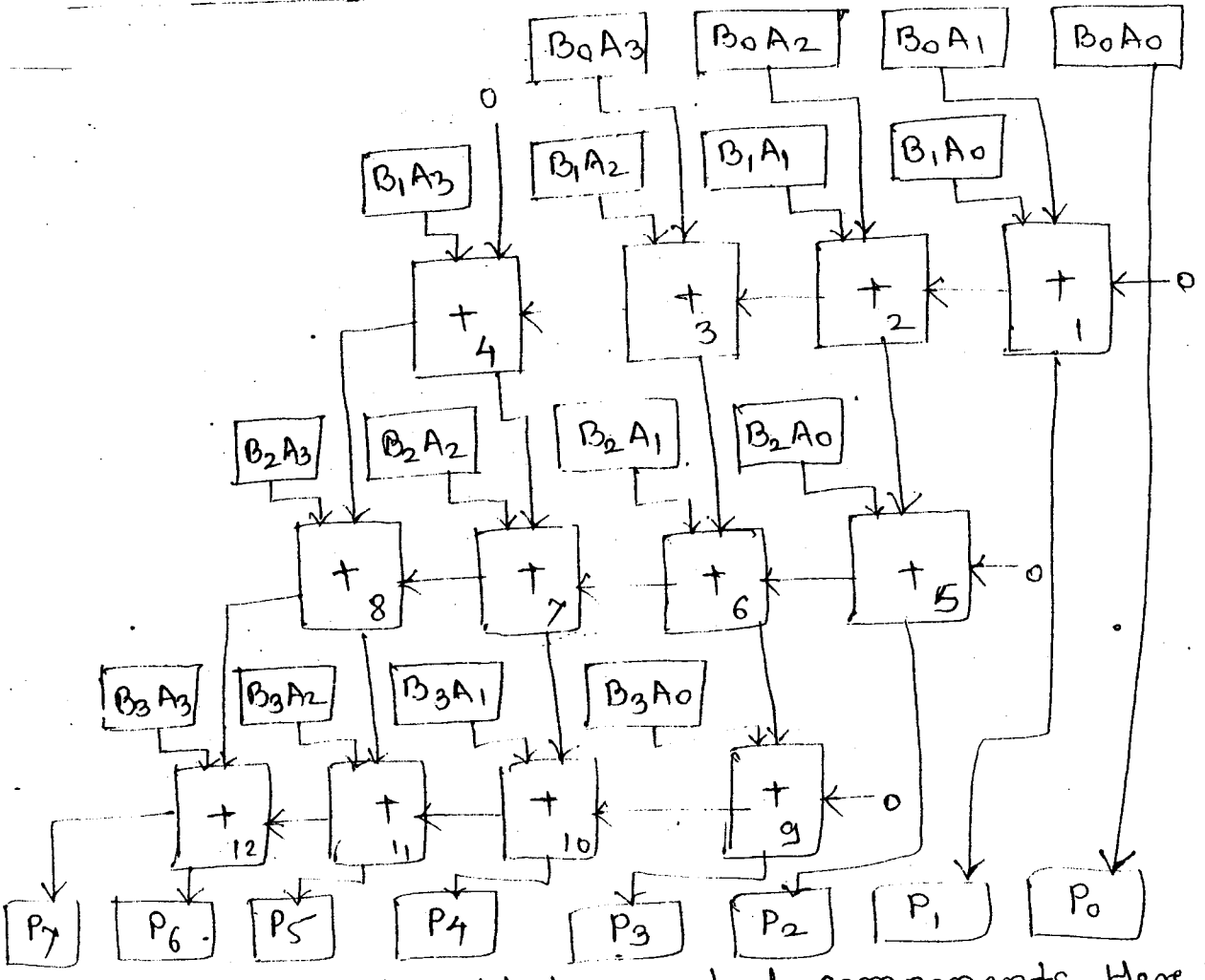
B_0A_3	B_0A_2	B_0A_1	B_0A_0
B_1A_3	B_1A_2	B_1A_1	B_1A_0
B_2A_3	B_2A_2	B_2A_1	B_2A_0
+ B_3A_3	B_3A_2	B_3A_1	B_3A_0

P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Multiplicand $A = A_3A_2A_1A_0$ &

such shifted multiplicand which is multiplied by ... depending on the corresponding multiplier bit is called partial product. Each partial product consists of four product component. The product component bit is a logical AND multiplier bit Bi & multiplicand bit Aj i.e $B_i \times A_j$. The final 8-bit product is obtained by adding all partial products.

* 4×4 Combinational Multiplier:

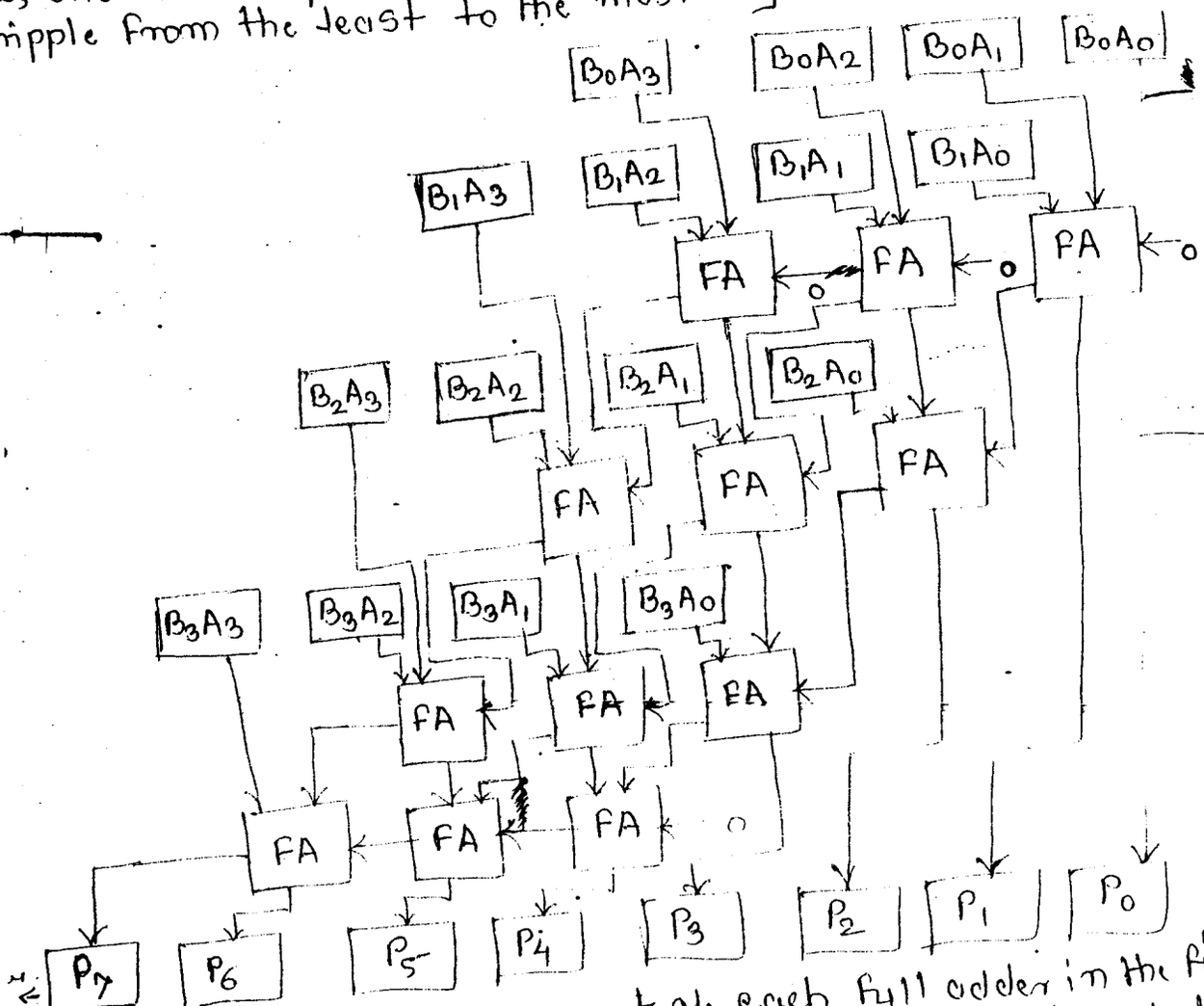


In this ckt to add the ~~product~~ components. Here, the product components are separated to make the space, & each '+' box is a full adder. The carrier in each partial product row of full adders are connected to make an 4-bit ripple adder. Thus, the first 4-bit ripple adder adds the first two rows of product component to produce the first partial product. The carry output generated is propagated to the most significant product component used to produce the next partial product. The subsequent adders adds each partial product with the next product component is obtained by adding all partial products.

Combinational Multiplier with Carry Save Addition:-

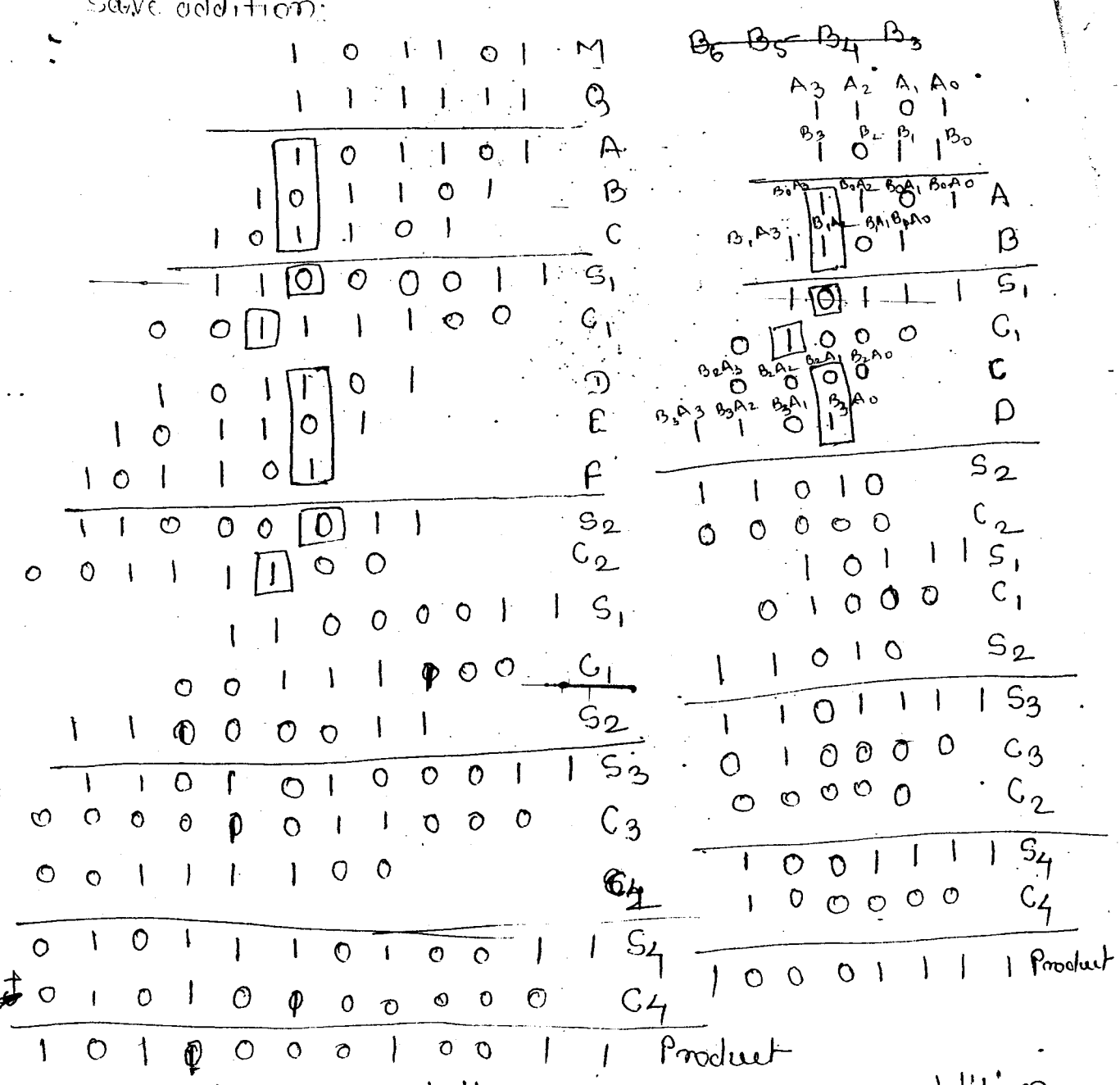
To increase the speed of the addition process many times technique called Carry-Save addition is used.

In this technique, the carry output from bit i during step j is applied to carry input for bit i+1 during the next step, j+1. After addition of product components in the last row, one more step is required in which the carries are allowed to ripple from the least to the most significant bit.

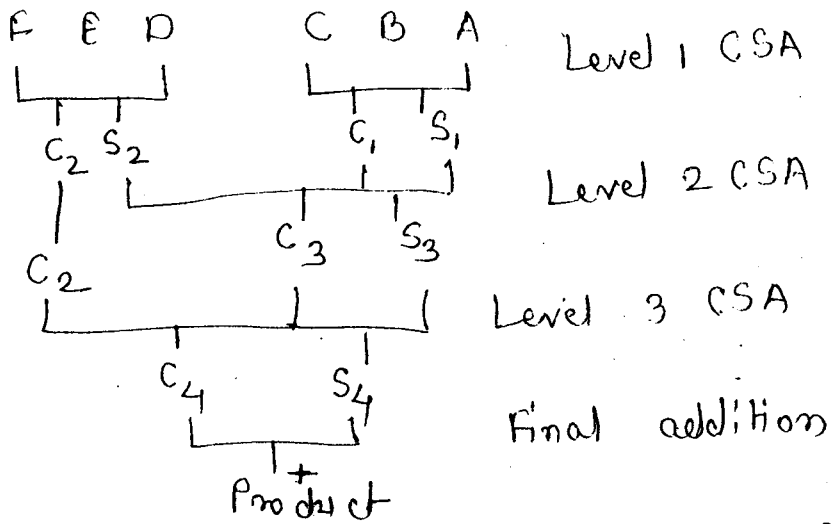


In this the carry out of each full adder in the first row of full-adders are connected to an input of an adder below it. Carries in the fourth row of full adders are connected to create a conventional ripple adder. This technique does not save any hardware but it reduces the propagation delay substantially.

ne multiplication example from performed using carry-save addition:



→ Schematic representation of the carry-save addition

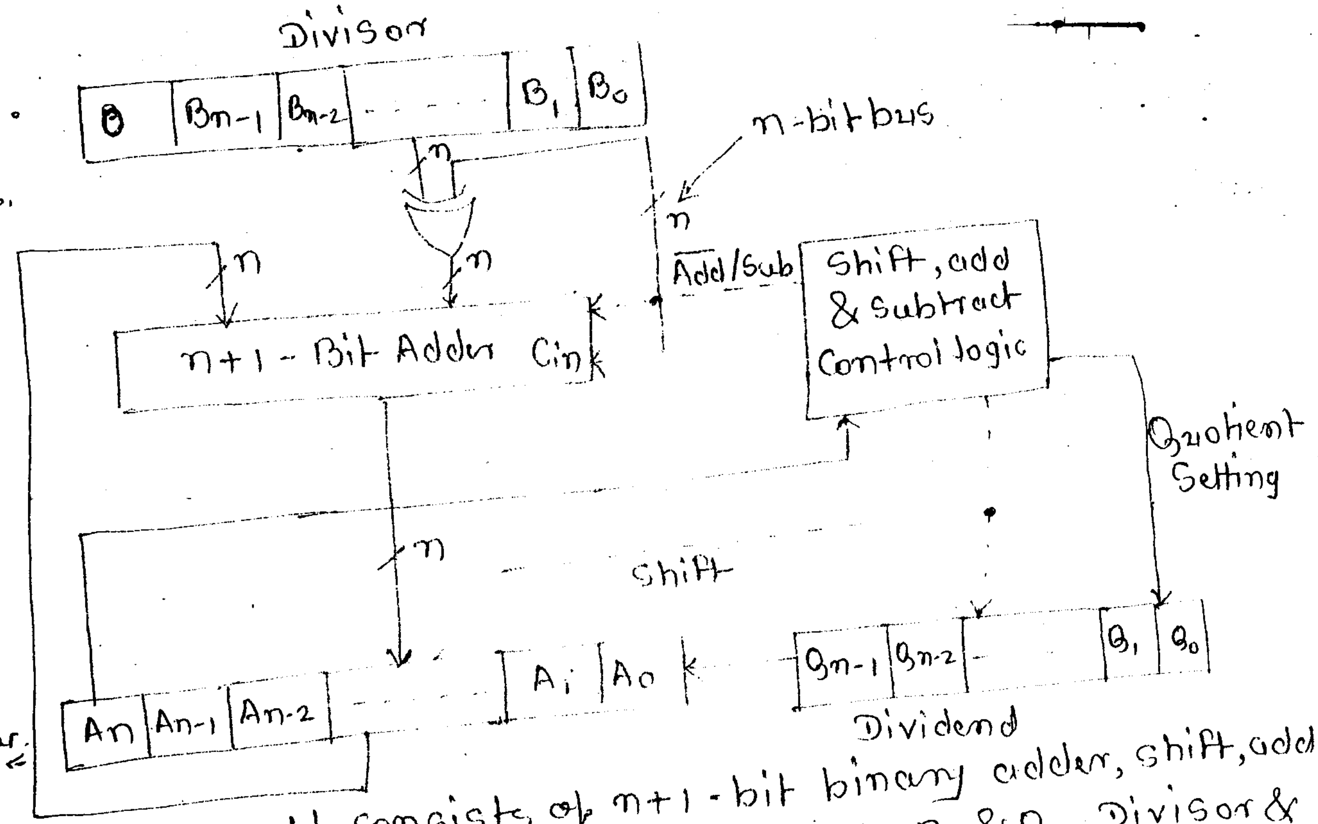


Division:

14 ← Quotient
12 (Divisor) | 169 ← Dividend
12
49 ← Partial Remainder
48
1

1110 ← Quotient
1100 (Divisor) | 10101001 ← Dividend
1100
10010
1100
00100
1100
00001 ← Remainder

* Implementation of ~~bit~~ division process:-



It consists of n+1-bit binary adder, shift, add & subtract control logic & registers A, B & Q. Divisor & dividend are loaded into register B & register Q. Register A is intially set to zero. The division operation is then carried out. After the division is complete, the n-bit quotient is in register Q & the remainder is in register A

* Division operation steps:- (Restoring division algorithm):

1. Shift A & Q left one binary position
2. Subtract divisor from A & place answer back in ($A \leftarrow A-B$)

... add divisor back to A

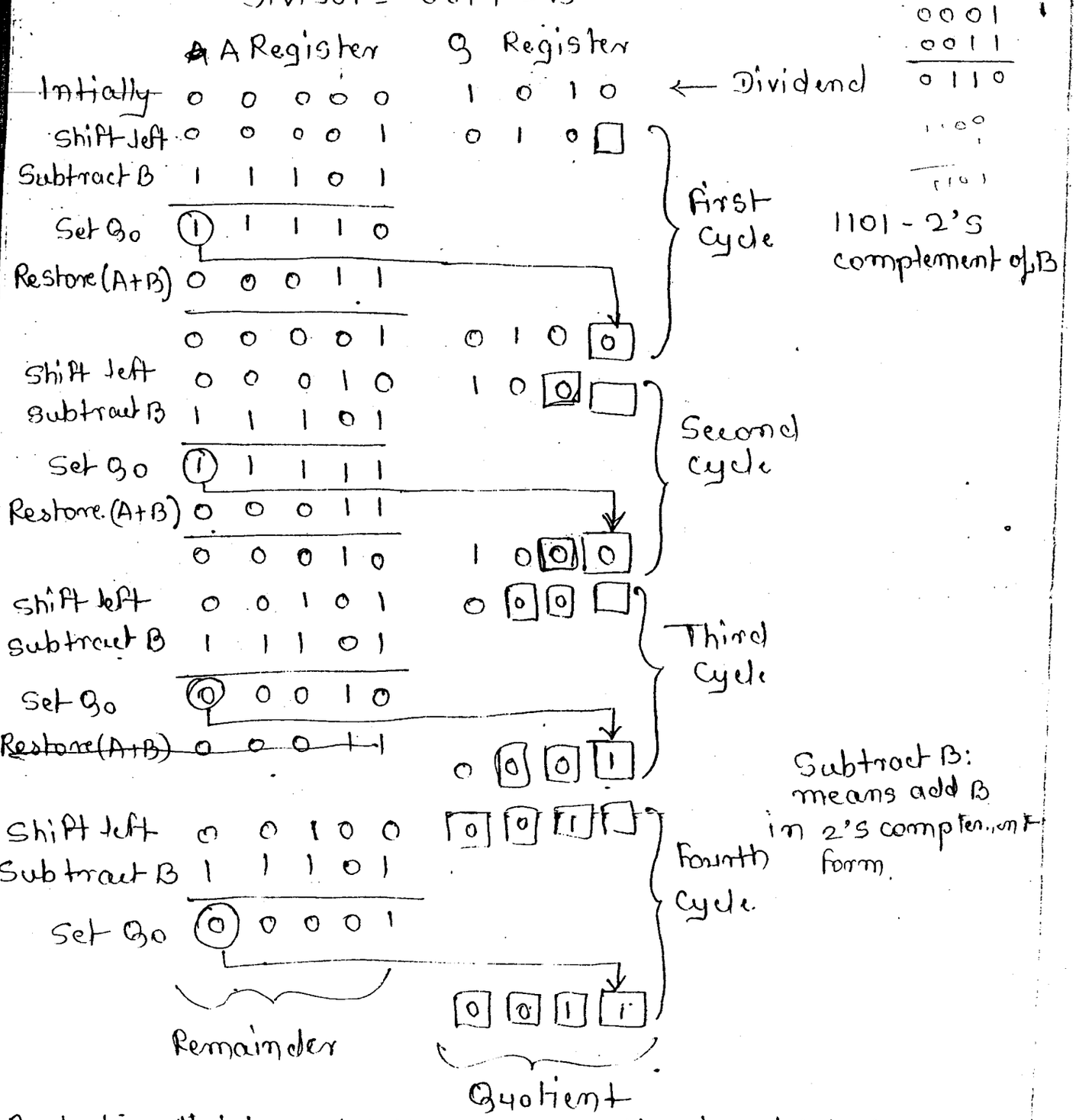
(that is restore A); otherwise, set Q_0 to 1.

4. Repeat steps 1, 2 & 3 n times.

for example: Consider 4-bit dividend & 2-bit divisor

Dividend = 1010 = Q

Divisor = 0011 = B



Restoring division algorithm: Subtraction is said to be unsuccessful if the result is negative (then restore A+B)

If A is positive: Shift left & Subtract divisor → 2A − B

If A is negative: Restore A + B
(Shift left & Subtract divisor → 2(A+B)-B
= 2A+B

* Non-restoring division algorithm:

Step 1: If the sign of A is 0, shift A & Q left one bit position & subtract divisor from A; otherwise shift A & Q left & add divisor to A.
If the sign of A is 0, set Q_0 to 1; otherwise set Q_0 to 0.

Step 2: Repeat steps 1 & 2 for n times.

Step 3: If the sign of A is 1, add divisor to A.

Note: ~~Step~~ 3 is required to leave the proper positive remainder in A at the end of n cycles.

for example: Dividend = 1010 & Divisor = 0011