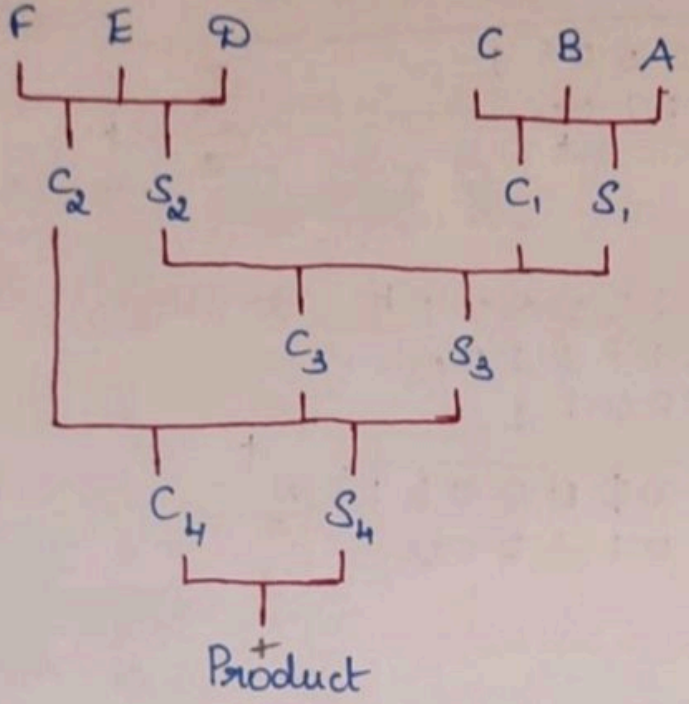
CARRY SAVE ADDITION OF SUMMANDS:

Algorithm:



Level 1 CSA

Level 2 CSA

Level 3 CSA

Final Adition

Example:

```
          1 0 1 1 0 1
        x 1 1 1 1 1 1
      -----------------
          1 0 1 1 0 1      A
        1 0 1 1 0 1 x      B
      1 0 1 1 0 1 x x      C
    1 0 1 1 0 1 x x x      D
  1 0 1 1 0 1 x x x x      E
1 0 1 1 0 1 x x x x x      F
```

```
          1 0 1 1 0 1
        1 0 1 1 0 1 x
      1 0 1 1 0 1 x x
    -----------------
      1 1 0 0 0 0 1 1      S1
    0 0 1 1 1 1 0 0        C1
```

```
  1 0 1 1 0 1 x x x
    1 0 1 1 0 1 x x x
  1 0 1 1 0 1 x x x x
-------------------------
  1 1 0 0 0 0 1 1
0 0 1 1 1 1 0 0
```

S_2
C_2

```
    1 1 0 0 0 0 1 1
  0 0 1 1 1 1 0 0
  1 1 0 0 0 0 1 1
-------------------------
  1 1 0 1 0 1 0 0 0 1 1
0 0 0 0 1 0 1 0 1 0 0 0
```

S_3
C_3

```
    1 1 0 1 0 1 0 0 0 1 1
  0 0 0 0 1 0 1 1 0 0 0
  0 0 1 1 1 1 1 0 0
-------------------------
  0 1 0 1 1 1 0 1 0 0 1 1
0 0 1 0 1 0 1 0 0 0 0 0
```

S_4
C_4

```
  0 1 0 1 1 1 0 1 0 0 1 1
0 0 1 0 1 0 1 0 0 0 0 0
-------------------------
0 1 0 1 1 0 0 0 1 0 0 1 1
```

Final Product.

Ans: 0101100010011

INTEGER DIVISION:

RESTORING METHOD:

Algorithm:

Step 1: $A \leftarrow 0$, $B \leftarrow$ Divisor, $Q \leftarrow$ Dividend, $n \leftarrow$ Count.

Step 2: Shift A and Q left one binary position.

Step 3: $A \leftarrow A - B$

Step 4: If A[n] is 1, assign $Q_0 \leftarrow 0$, $A \leftarrow A + B$ (Restore)

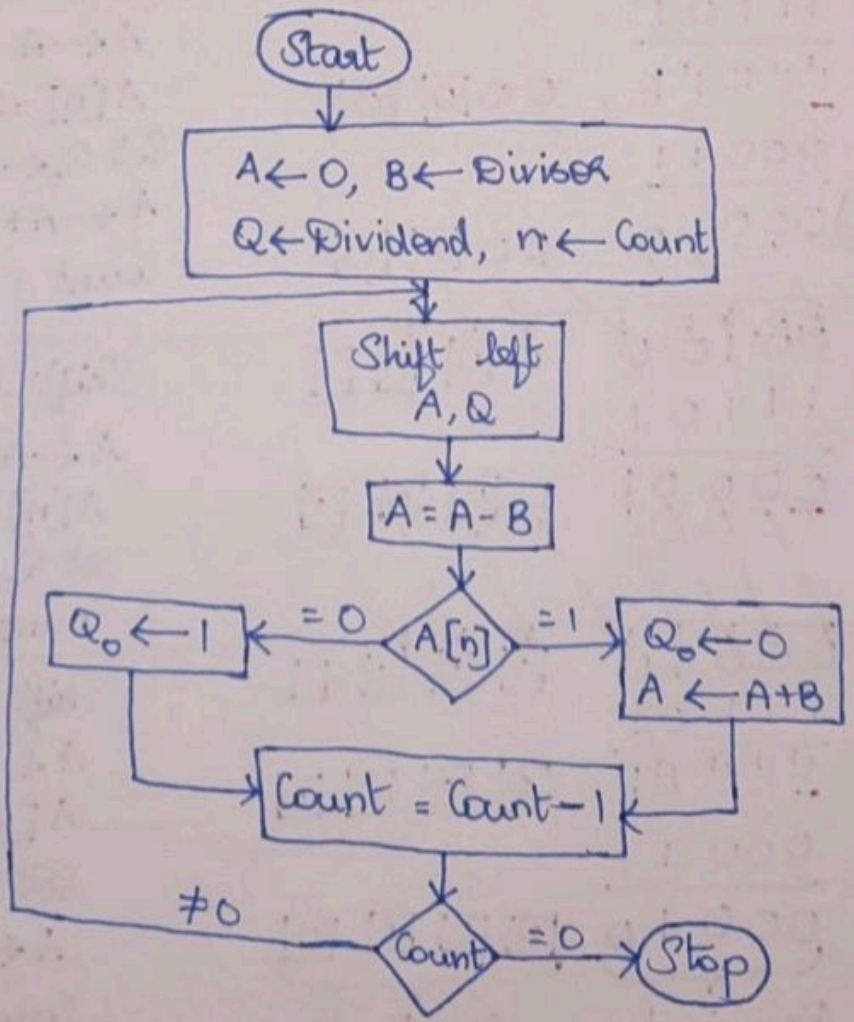
If A[n] is 0, assign $Q_0 \leftarrow 1$.

Step 5: Count $\leftarrow$ Count - 1

Step 6: Repeat steps 2, 3, 4, 5 until Count become zero.

Step 7: If Count = 0, then stop. Else continue the process.

Flow Chart:



Example:

Divide 8 by 3

$8 \div 3 \Rightarrow 1000 \div 11$

$Q = 1000$

$B = 0011$

Count = 4

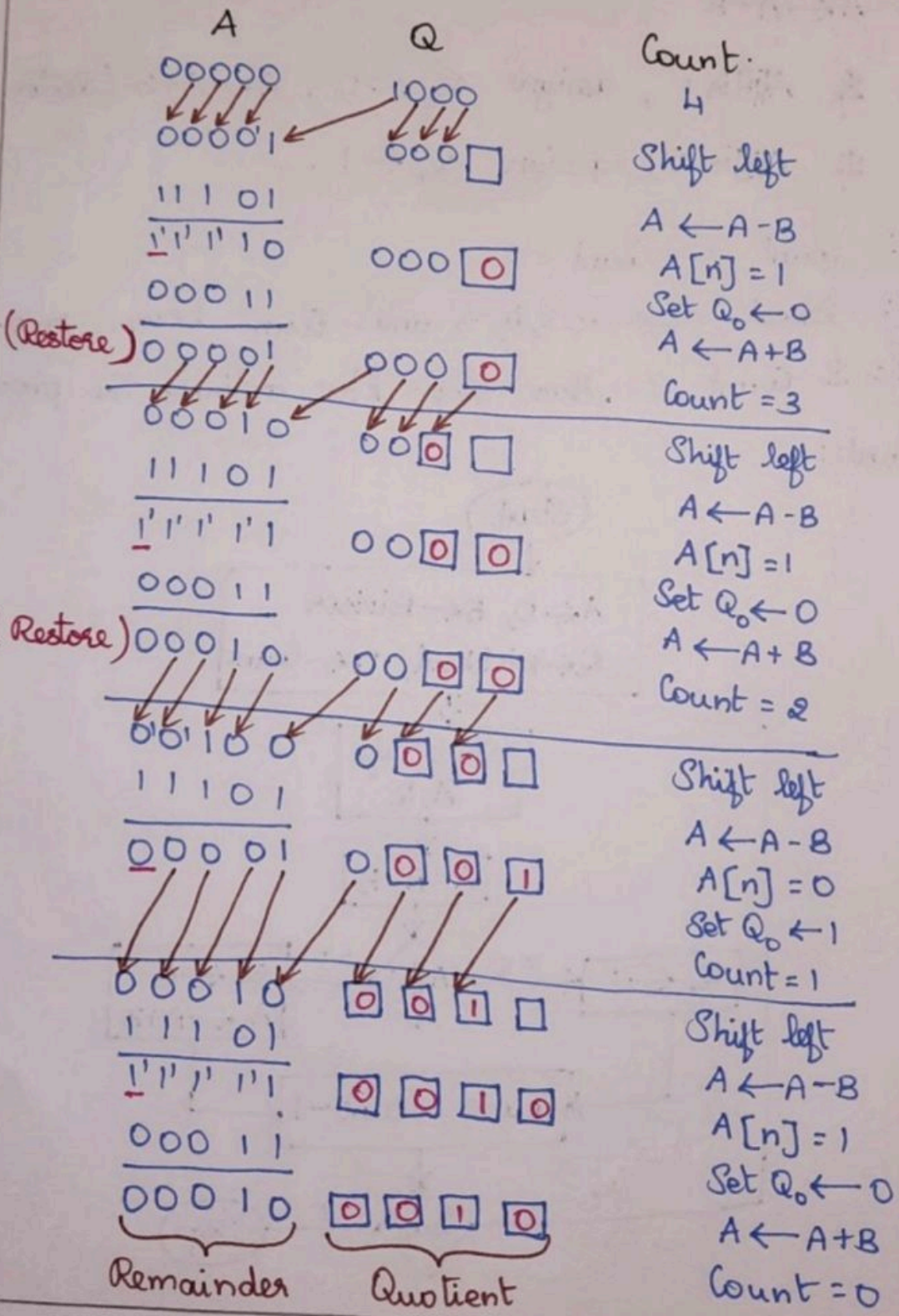
$B = 0011$

$B = 00011$

1's com $= 11100$

$+1$

$-B = 11101$



NON-RESTORING METHOD:

Algorithm:

Step 1: A ← 0, B ← Divisor, Q ← Dividend, n ← Count.

Step 2: Shift left A, Q one binary position

Step 3: If A[n] is 1, A ← A+B

If A[n] is 0, A ← A-B

Step 4: If A[n] is 1, $Q_0 \leftarrow 0$

If A[n] is 0, $Q_0 \leftarrow 1$

Step 5: Count = Count - 1

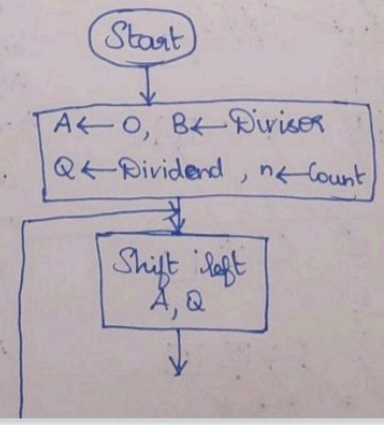
Step 6: If Count ≠0, repeat steps 2, 3, 4, 5 until Count becomes zero.

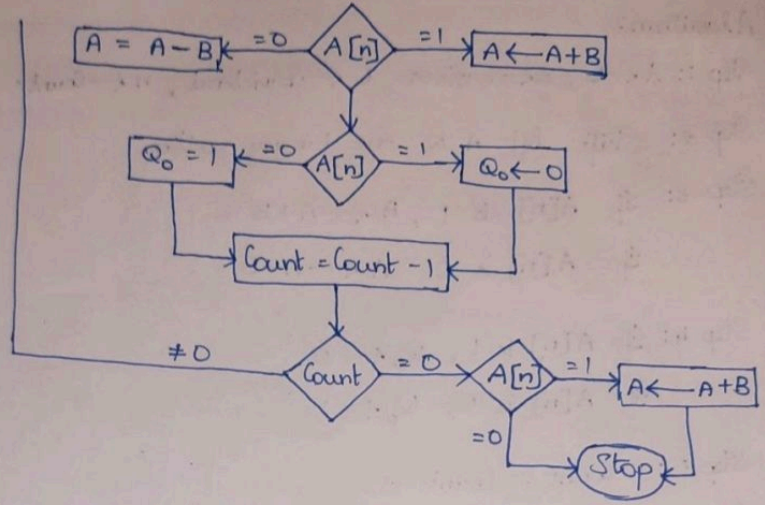
Step 7: If count =0, check A[n] bit.

Step 8: If A[n] is 1, A ← A+B

If A[n] is 0, stop.

Flow Chart:





Example:

Divide 8 by 3.

$8 \div 3 \Rightarrow 1000 \div 11$

Q = 1000

B = 00011

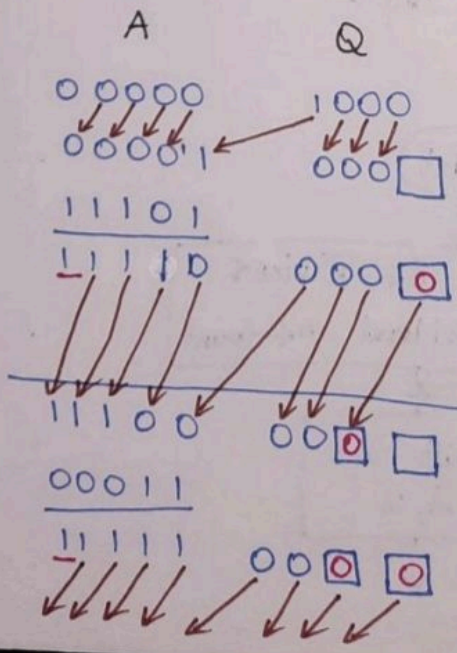
Count = 4

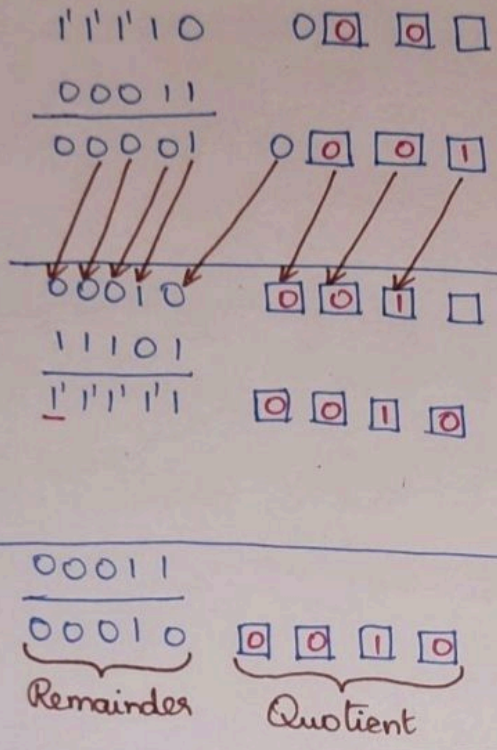
B = 00011

1's com = 11100

+1

$-B = 11101$





BIT PAIR RECODING: FAST MULTIPLICATION:

*. Supports fast multiplication
*. Reduced number of steps compared to Booth's Algorithm.
*. 3 bits comparison.
*. Speeds up multiplication process.

MULTIPLIER		MULTIPLIER	BIT PAIR
i+1	i	i-1	RECODER
0	0	0	0 x M
0	0	1	+1 x M
0	1	0	+1 x M
0	1	1	+2 x M
1	0	0	-2 x M
1	0	1	-1 x M
1	1	0	-1 x M
1	1	1	0 x M

Example:

Multiply: 27 & -11

Multiplicant ⇒ 110101 (-11)
Multiplier ⇒ 011011 (27)

Recode Multiplier:

0 1 1 0 1 1 [0
+2 -1 -1

-1 ⇒ 2's comp. of multiplicant
+2 ⇒ Multiplicant x 10
-2 ⇒ 2's comp. of multiplicant x 10
+1 ⇒ Normal.

```
          1 1 0 1 0 1
          +2  -1  -1
        ---------------
          0 0 1 0 1 1
        0 0 1 0 1 1 x x
      1 1 0 1 0 1 0 x x x x
    -------------------------
      1 1 0 1 1 0 1 0 1 1
    -------------------------
```

```
          1 1 0 1 0 1
1's c ⇒   0 0 1 0 1 0
                  + 1
          -----------
2's c ⇒   0 0 1 0 1 1
          -----------
```

110101 x 10
= 1101010

Hazards

- prevent the next instruction in the instruction stream from being executing during its designated clock cycle

- Three classes of hazards:
  - Structural hazards
  - Data hazards
  - Control hazards

Structural Hazard

- arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution

Common instances of structural hazards arise when :

- Some functional unit is not fully pipelined - then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle

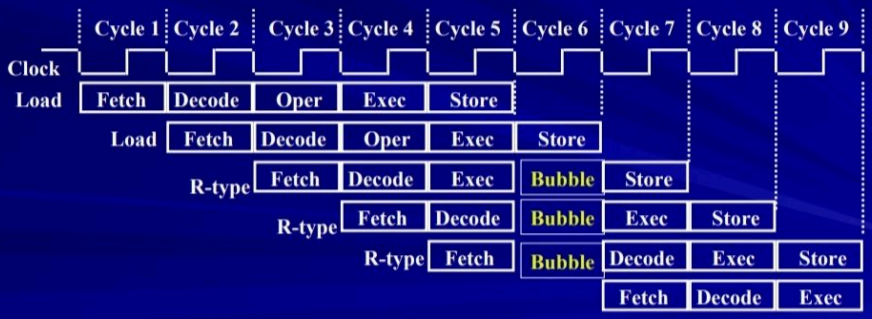
- Some resource has not been duplicated enough - to allow all combinations of instructions in the pipeline to execute

Solution of Structural Hazard

- stall the pipeline for one clock cycle when a data-memory access occurs (bubbles)
- Stall - prevent the succeeding instruction from doing its phase of the cycle
- allows the stalled instruction to proceed without conflict but defeats the purpose of overlapping the instructions for the stage

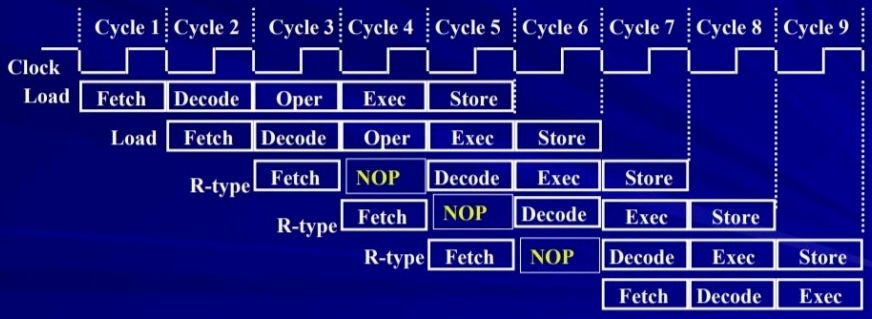
Pipeline Bubble

Insert a **bubble** into the pipeline to prevent two writes or stores at the same cycle



Delay Store Cycle

Add NOP (No Operation)



Data Hazard

- arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline

Data Hazard Classification

Consider two instructions ***i*** and ***j***, with ***i*** occurring before ***j***. The possible data hazards are :

- ***RAW (read after write)*** - ***j*** *tries to read a source before* ***i*** *writes it, so* ***j*** *incorrectly gets the old value*

- Solution: Forwarding ("Forward" result from one stage to another)
  - The ALU result from the Op Fetch /Exec register is always fed back to the ALU input latches
  - If the forwarding hardware detects that the previous ALU operation has written the register corresponding to the source for the current ALU operation, ***control logic*** selects the forwarded result as the ALU input rather than the value read from the register file

Data Hazard Classification

Consider two instructions ***i*** and ***j***, with ***i*** occurring before ***j***. The possible data hazards are :

- ***WAW (write after write)*** - ***j*** *tries to write an operand before it is written by* ***i****. The writes end up being performed in the wrong order, leaving the value written by* ***i*** *rather than the value written by* ***j*** *in the destination*
- Solution: Stall

Data Hazard Classification

Consider two instructions ***i*** and ***j***, with ***i*** occurring before ***j***. The possible data hazards are :

- ***WAR (write after read)*** - ***j*** *tries to write a destination before it is read by* ***i*** *, so* ***i*** *incorrectly gets the new value*
- Solution: Stall

Control Hazard

- arise from the pipelining of branches and other instructions that change the PC
- Solution: Stall the pipeline as soon as the branch is detected - Program Counter value changed to target address
  - Compute target address in advance