COA – MODULE 2

IEEE 754 Standard For Floating Point Representation

* Floating Point Representation :-

$(5.625)_{10} \longrightarrow (101.101)_2$

0.5625×10^{1}

Mantissa — Exponent

0.101101×2^{3}

Mantissa — Exponent

→ In computer, we used a fixed bit memory space for representation

* Normalization :-

→ Need of Normalization

$(101.101)_2 \longrightarrow (0.101101 \times 2^3)$

$(101.101)_2 \longrightarrow (101101 \times 2^{-3})$

$(101.101)_2 \longrightarrow (1.01101 \times 2^2)$

∴ Various floating point representation is possible. So, standarization is required. This process called Normalization

Normalization

Explicit

→ Move radix point to the LHS of the most significant '1' in the bit sequence.

$(101.101)_2 \rightarrow (0.101101 \times 2^3)$

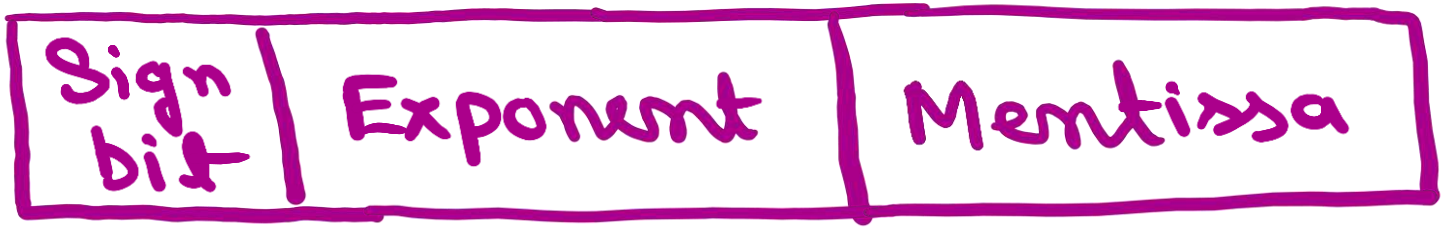
Implicit

→ Move the radix point to the RHS of the most significant '1' in the bit sequence

$(101.101) \rightarrow (1.01101 \times 2^2)$

* Biasing :-

→ To store a number, we required a fixed bit storing element like below



$(101.101)_2 \longrightarrow 101101 \times 2^{-3}$ → exponent can also -ve & +ve for different representation

Now, question is how to represent those -ve number → 2's compliment?

→ In, 2's compliment representation of n-bit number, the range is $\left[-2^{n-1} \text{ to } +(2^{n-1}-1)\right]$.

→ For n=4, range of 2's comp. representation,

$$\left[-2^{4-1} \text{ to } +(2^{4-1}-1)\right] \Rightarrow [-8 \text{ to } 7]$$

$$\Downarrow$$

$$-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7$$

We need to convert it into unsigned representation

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
+8	+8	+8	+8	+8	+8	+8	+8	+8	+8	+8	+8	+8	+8	+8	+8
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

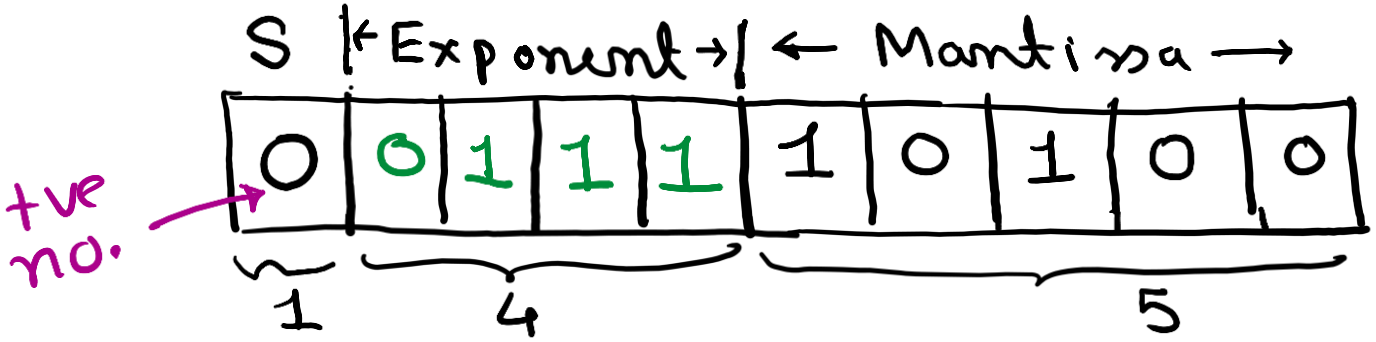
Add 8 ↓ excess 8

Ex:-

i) $(0.0101)_2 \longrightarrow 0.1101 \times 2^{-1}$ ⇒ Explicit Representation

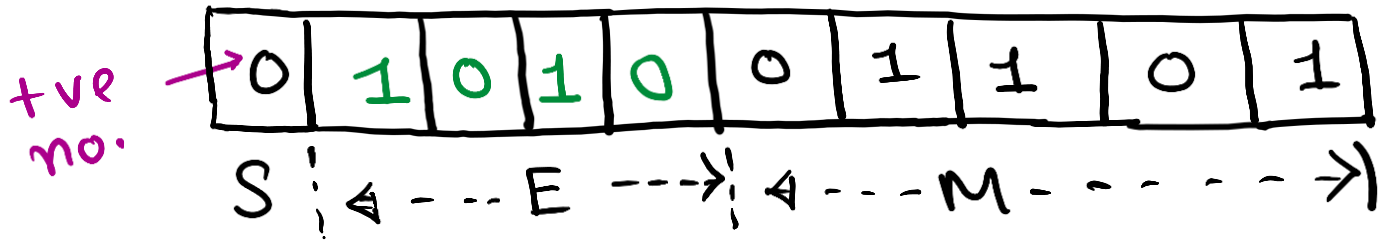
7 ⇒ (0111)

(10 bit) memory space.



ii) $(101.101)_2 \longrightarrow (1.01101 \times 2^{+2})$ ⇒ Implicit Representation

10 ⇒ $(1010)_2$



⇐ 10 bit memory space

→ To convert the memory stored number into human readable form,

1. Explicit Normalization : $(-1)^{S} \times 0.M \times 2^{E\text{-bias}}$

2. Implicit Normalization :- $(-1)^{S} \times 1.M \times 2^{E\text{-bias}}$

IEEE 754 Binary Floating Point Standard :-

Name	Common Name	Significant Bits (M+S)	Exponent Bits	Exponent Bias	S, M, E
Binary16	Half Precision	11	5	15 (excess 15)	S → 1 M → 10 E → 5
Binary 32	Single Precision	24	8	127 (excess 127)	S → 1 M → 23 E → 8
Binary 64	Double Precision	53	11	1023 (excess 1023)	S → 1, M → 52 E → 11
Binary 128	Quadruple Precision	113	15	16383 (excess 16383)	S → 1, M → 112 E → 15
Binary 256	Octuple Precision	237	19	262143 (excess 262143)	S → 1, M → 236 E → 19

IEEE 754 - Single Precision (Binary 32)

S (1 bit)	E (8 bit)	M (23 bit)	Representation
0 or 1	0000 0000 $E = (0)_{10}$	000000 ....0000 $(M = 0)_{10}$	± 0
0 or 1	1111 1111 $E = (255)_{10}$	00000 .... 0 $(M = 0)_{10}$	$\pm \infty$
0 or 1	$1 \leq E \leq 254$	$M = x x x x \ldots x$	Implicit Normalized Form $(-1)^S \times 1.M \times 2^{E-127}$
0 or 1	$E = (0)_{10}$	$M \neq 0$	Denormalized or Fractional Form :- $(-1)^S \times 0.M \times 2^{-126}$
0 or 1	$E = (255)_{10}$	$M \neq 0$	NAN (not a number) e.g:- %; overflow.

IEEE 754 - Double Precision (Binary 32)

S (1 bit)	E (11 bit)	M (52 bit)	Representation
0 or 1	00000000000 $E = (0)_{10}$	000000000 ... 0 $M = (0)_{10}$	± 0
0 or 1	11111111111 $E = (255)_{10}$	000000000 ... 0 $M = (0)_{10}$	$\pm \infty$
0 or 1	$1 \leq E \leq 2046$	$M = x x x x \ldots x$	Implicit Normalized Form $(-1)^S \times 1.M \times 2^{E-1023}$
0 or 1	$E = (0)_{10}$	$M \neq 0$	Denormalized or Fractional Form :- $(-1)^S \times 0.M \times 2^{-1022}$
0 or 1	$E = (2047)_{10}$	$M \neq 0$	NAN (not a number) e.g:- %, overflow.

* Suppose a 16 bit register of the following format is used for storing binary point numbers :-

[Mantissa (M) is denoted using normalized sign-magnitude fraction. Exponent (E) is expressed in excess 64 form]

1. How many bits used for the fractional mantissa.
2. What should be the expression for the decimal value?
3. What is largest number (in Base 10) that can be represented using this register.
4. What should be the bit patterns for $(7.5)_{10}$ & $(-16.125)_{10}$?
5. Determine the difference between the smallest +ve number and 2nd smallest positive number in the presentation.

1) Excess 64 is used,

$\therefore$ Maximum -ve value $= -64 = -2^6 = -2^{-7-1}$

↳ 2's Comp. lowest range.

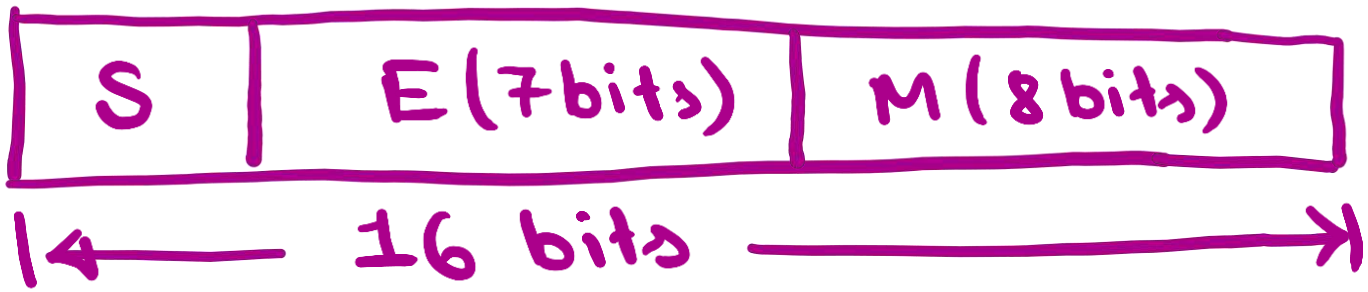
$\left[-2^{n-1} \text{ to } +(2^{n-1}-1)\right]$

↑ Max -ve value

$\therefore$ Exponent bits(E) = 7

$\therefore$ Sign bit (S) = 1

$\therefore$ Mantissa (M) = 8 bit



} Memory Space

2) As nothing is mentioned, so we assume implicit normalization because it gives better precision over explicit normalization.

∴ Implicit Normalization Form (Human Readable)

$$(-1)^S \times 1.M \times 2^{E-bias}$$

∴ For excess bias -64

$$(-1)^S \times 1.M \times 2^{E-64} \Longrightarrow$$

Expression for the decimal value.

8) The largest number (in base 10) that can be represented using this register :-

Max value of Mantissa

$(11111111)_2$

Max value of exponent

$(11111111)_2$

$(2^7 - 1)_{10} = (128 - 1)_{10}$

$= (127)_{10}$

For max value sign bit must be +ve ; S = 0

→ Implicit Normalization,

E
127-64
↳ Bias

0 → S

Value ⇒ $(-1) \times 1.\underbrace{11111111}_{M} \times 2$

⇒ $(1.11111111 \times 2^{63})_2$

⇒ $(1111\ 1111 \times 2^{55})_2$ (Shift radix point RS)

⇒ $(1111\ 1111\ 1000\ 0000 \times 2^{50})_2$ → For exponent

Converting into Hexa.

Face values of Hexadecimal

0011 → 3 | F(15) | E(14) | 0

$(3FE0)_{16}$

$2^{10} \cong 10^3$

$2^{50} \cong 10^{15}$

∴ Largest decimal number can store inside of 16 bit register.

$$\text{Value} \Rightarrow \left(3 \times 16^3 + 15 \times 16^2 + 14 \times 16^1 + 0 \times 16^0\right) \times 10^{15}$$

$$\Rightarrow \left(16352 \times 10^{15}\right)_{10} \quad \cdots (\text{Ans})$$

4:

a) Bit Pattern of $(7.5)_{10}$:-

Implicit form

$(7.5)_{10} \rightarrow (111.1)_2 = 1.111 \times 2^{2}$ → E

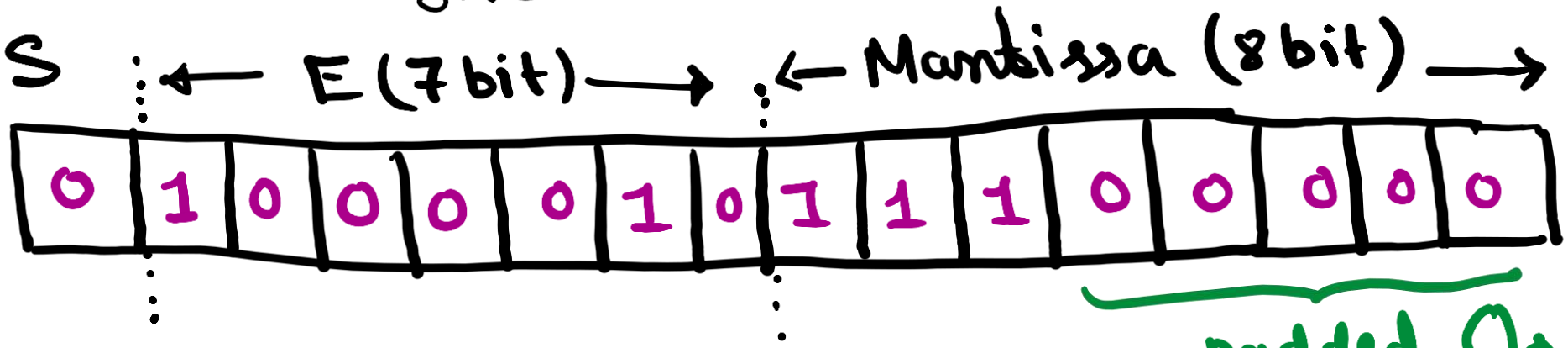
M

S = 0 ⇒ +ve no.

7 bit

$E = 2 + 64 = (66)_{10} = (1000010)_2$

excess 64 given



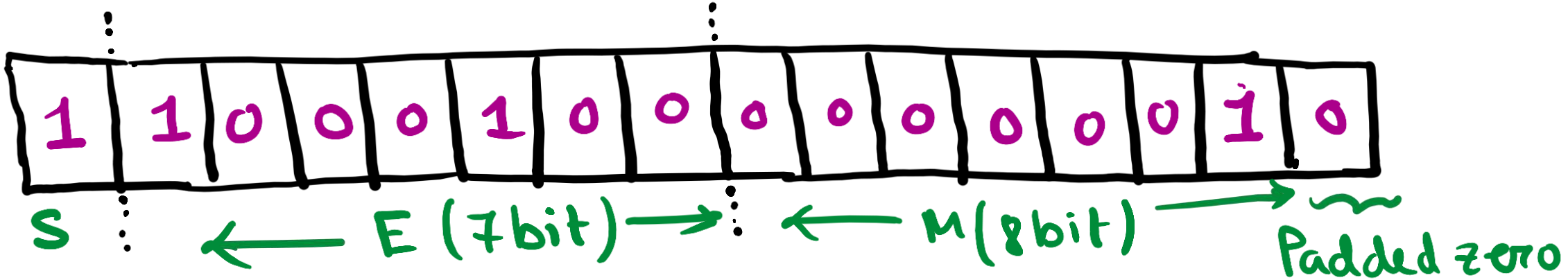
4. b) Bit pattern of $(-16.125)_{10}$:-

$(-16.125)_{10} \longrightarrow (10000.001)_2$

$\Rightarrow 1.\underbrace{0000001}_{M} \times 2^{4}$

$S = 1 \Rightarrow$ -ve no.

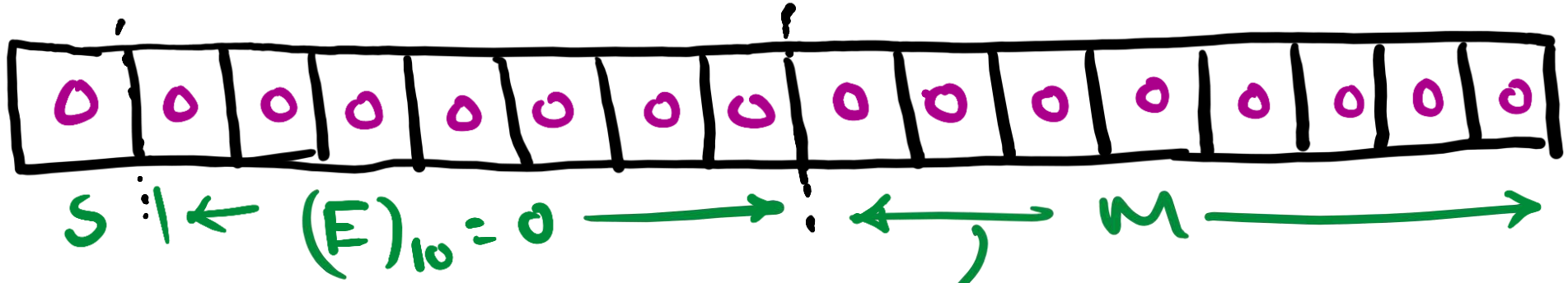
$E = 4 + 64 = (68)_{10} = (1000100)_2$



5. Smallest (+ve) Number :-

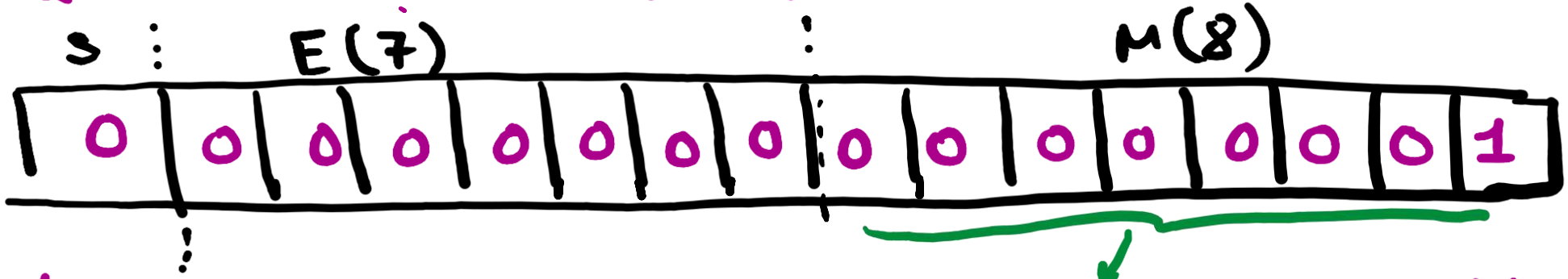
→ Let take a implicit normalization form with excess 64 bias,

$$(-1)^S \times 1.M \times 2^{E-64}$$



Smallest Value => $(-1)^0 \times 1.00000000 \times 2^{0-64} = 1.0 \times 10^{-64}$

→ 2nd Samallest Number :-



2nd Smallest Value $\Rightarrow (-1)^0 \times 1.00000001 \times 2^{0-64}$

$\Rightarrow 1.00000001 \times 2^{-64}$

Smallest +ve no. :- $1.00000000 \times 2^{-64}$

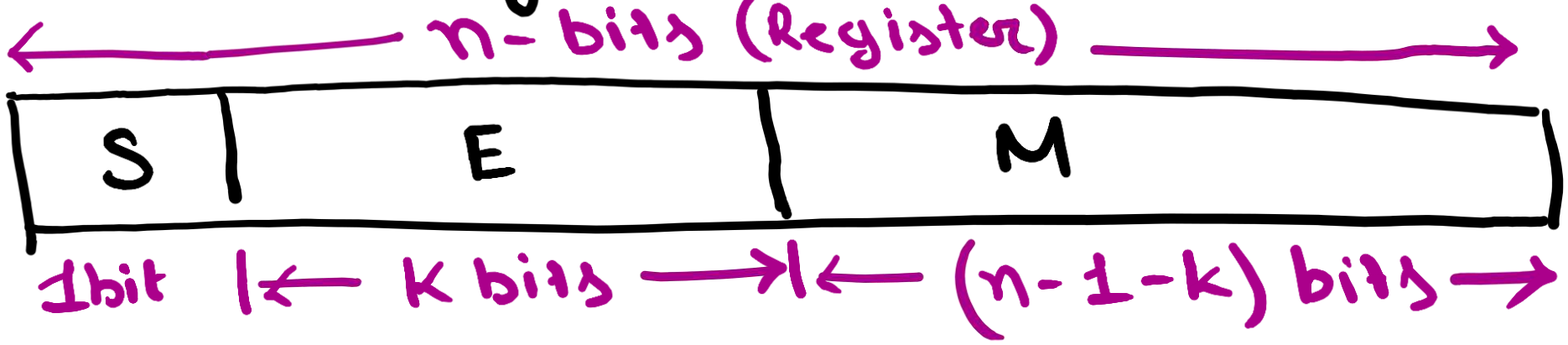
2nd Smallest +ve no. :- $1.00000001 \times 2^{-64}$

Difference :- $0.00000001 \times 2^{-64}$

Shift Radix Point →

$\Rightarrow 1 \times 2^{-8} \times 2^{-64} \Rightarrow 1 \times 2^{-72}$ ... (A)

Note:: Floating Point Numbers (Generally)



Baised exponent : $0 \leq E \leq 2^{k-1}$

Bias : $(2^k - 1)$

∴ True Exponent = (Biased Exponent - Bias)

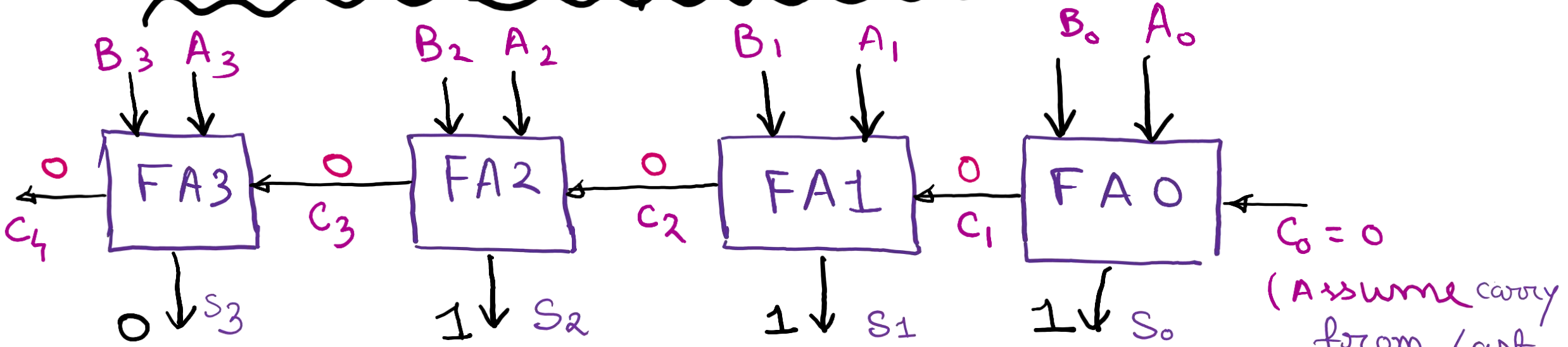
Design of Adder

→ Ripple Carry Adder

→ Carry Look Ahead Adder

Note:: Both are comes under "Parallel Adder"

1. Ripple Carry Adder :-



$A_3\ A_2\ A_1\ A_0 \longrightarrow$ 1 0 1 0

$B_3\ B_2\ B_1\ B_0 \longrightarrow$ 1 1 0 1

1 0 0 0 0

1 0 1 1 1

Carry C_4

$t_p \Rightarrow$ **Propagation Delay**

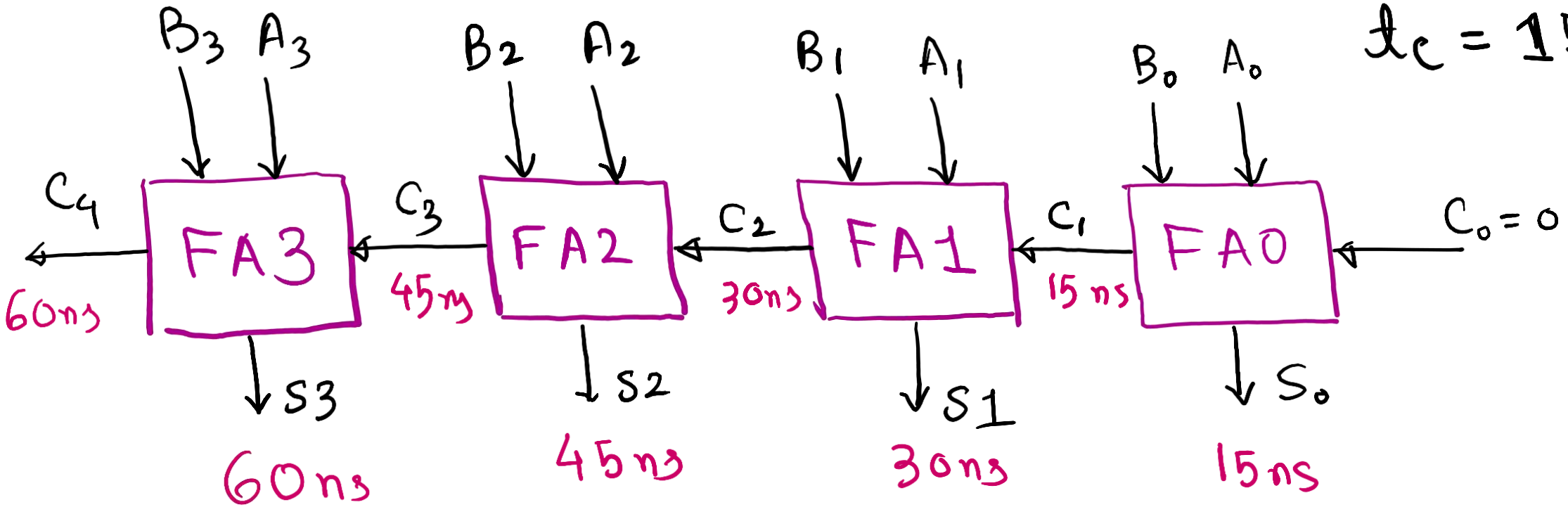
$\Rightarrow$ Time required to get output in a stage.

$\therefore$ t_p can be divided into two parts

- t_s: Propagation delay for Sum output.
- t_c:- Propagation delay for Carry output.

→ Let consider a 4 bit adder with $t_s = 15ns$

$t_c = 15ns$



$(3t_c + t_s)$

∴ if $t_s \neq t_c$, then for n-bit adder

- Total Propagation Delay for Sum output $= (n-1)t_c + t_s$

- Total Propagation Delay for Carry output $= nt_c$

Note:-

→ The carry propagation delay of the ripple carry adder is the main disadvantage. ⇒ To resolve it

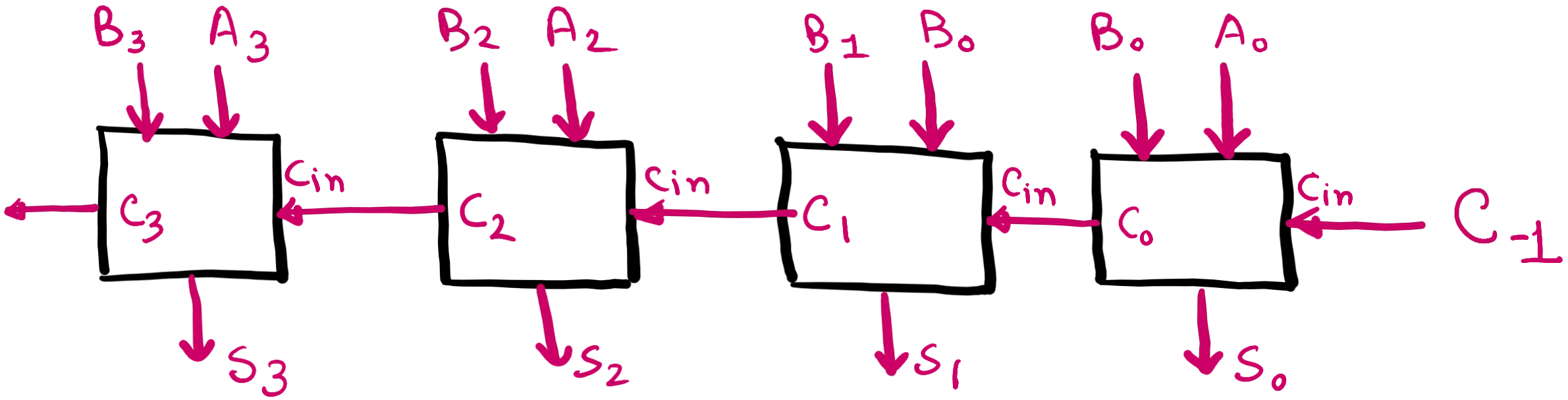
↓

We go for ⇒ "Look Ahead Carry Adder"

Carry Look Ahead Adder (CLA):-

→ Let predict carry of next stage before it will generate

→ Let consider a 4-bit parallel adder :-



A_i	B_i	C_{in}	C_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$A_i \oplus B_i = 1$
&
$C_{in} = 1$

when
$A_i = 1$
$B_i = 1$
$\rightarrow C_i = 1$

→ Logic for carry predictor,

$$C_i = \underbrace{A_i B_i}_{\text{Carry Generator } (G_i)} + \underbrace{(A \oplus B)}_{\text{Carry Propagator } (P_i)} \cdot C_{in}$$

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

where $i = 0, 1, 2, \ldots$

$i=0$

$$C_0 = G_0 + P_0 C_{-1} \quad \cdots (i)$$

$i=1$

$$C_1 = G_1 + P_1 C_0$$

$$C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{-1}$$

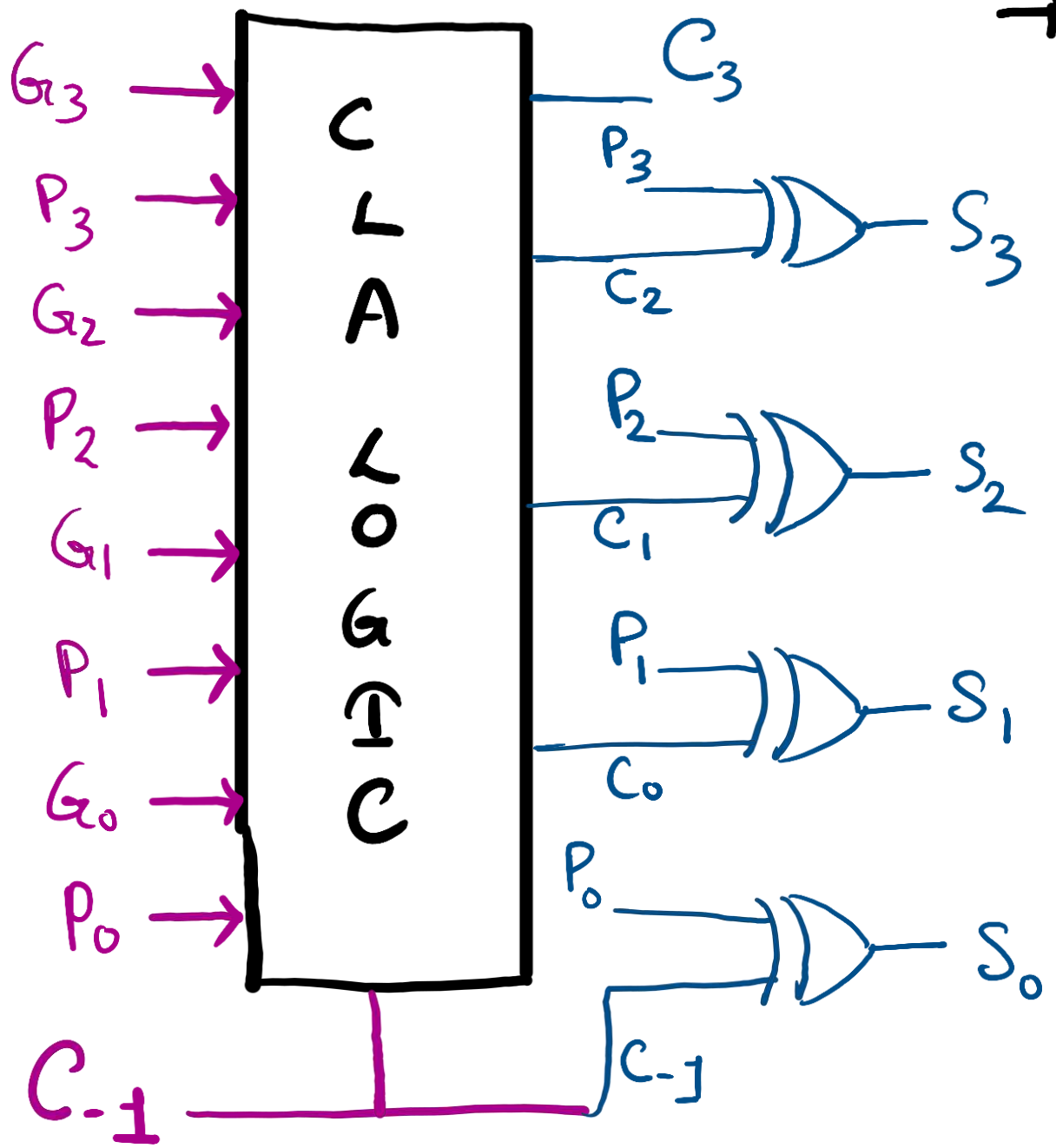
$i=2$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1}$$

$i=3$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1}$$

* if we know C_{-1} (initial value), then we can easily predict C_1, C_2 in advance.



→ 4bit CLA, carry predictor

$G_0 = A_0 B_0$, $G_1 = A_1 B_1$

$G_2 = A_2 B_2$, $G_3 = A_3 B_3$

$P_0 = A_0 \oplus B_0$, $P_1 = A_1 \oplus B_1$

$P_2 = A_2 \oplus B_2$, $P_3 = A_3 \oplus B_3$

$S_3 = P_3 \oplus C_2$

$S_2 = P_2 \oplus C_1$

$S_1 = P_1 \oplus C_0$

$S_0 = P_0 \oplus C_{-1}$

Densign of Multiplier

- → Shift and add method
- → Booth Multiplier
- → Carry Save Multiplier (Home Assignment)

* Shift and Add Method:-

→ Pen and Paper Method

e.g:- (14) × (10) = 140 ⇐ decimal

14 ⟶ 1 1 1 0 ⇒ (M:- Multiplicand)

10 ⟶ 1 0 1 0 ⇒ (Q:- Multiplier)

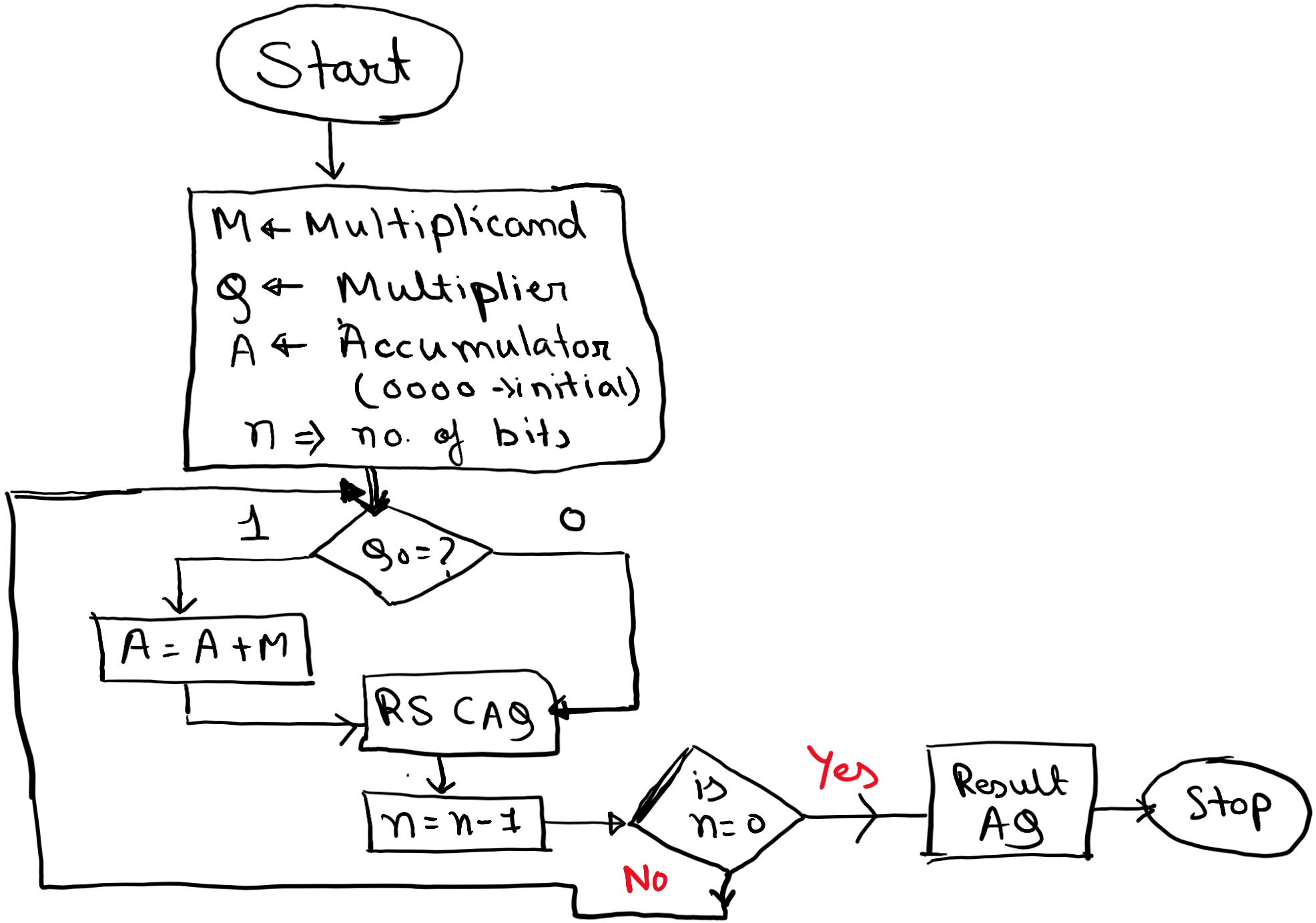
```
          0 0 0 0
        1 1 1 0
      0 0 0 0
    1 1 1 0
-------------------
(1 0 0 0 1 1 0 0)2  ⟶  (140)10
```

} Partial Product

Let, $\underbrace{11}_{M} \times \underbrace{13}_{9} = (143)_{10}$ → Final Ans.

$(1011) \times (1101) = (\ ?\ )_2$

$= (1000\ 1111)_2$


Start
M ← Multiplicand
Q ← Multiplier
A ← Accumulator
(0000 → initial)
n ⇒ no. of bits
1
0
Q_0 = ?
A = A + M
RS CAQ
n = n - 1
is n = 0
Yes
No
Result AQ
Stop

n	M	C	A	Q	Operation
4	1011	0	0000	$Q_3\ Q_2\ Q_1\ Q_0$ 1101 ✓	Initialization
	Carry of A+M →	0	1011	1101	If $Q_0=1$, then A = A+M
		0	0101	1110	Shift Right C A Q
3	1011	0	0010	1111	If $Q_0=0$, then Shift Right C A Q
2	1011	0	1101	1111	If $Q_0=1$, then A=A+M
		0	0110	1111	Shift Right C A Q
1	1011	1	0001	1111	$Q_0=1$, then A=A+M
		0	1000	1111	Shift Right C A Q

$$AQ = (\overset{2^7}{1}\ \overset{2^6}{0}\ \overset{2^5}{0}\ \overset{2^4}{0}\ \ \overset{2^3}{1}\ \overset{2^2}{1}\ \overset{2^1}{1}\ \overset{2^0}{1})_2$$

$$\Downarrow$$

$$= (128 + 15)_{10}$$

$$\boxed{AQ = (143)_{10}}$$

M → 4bit or n-bit

Q → 4bit or n-bit

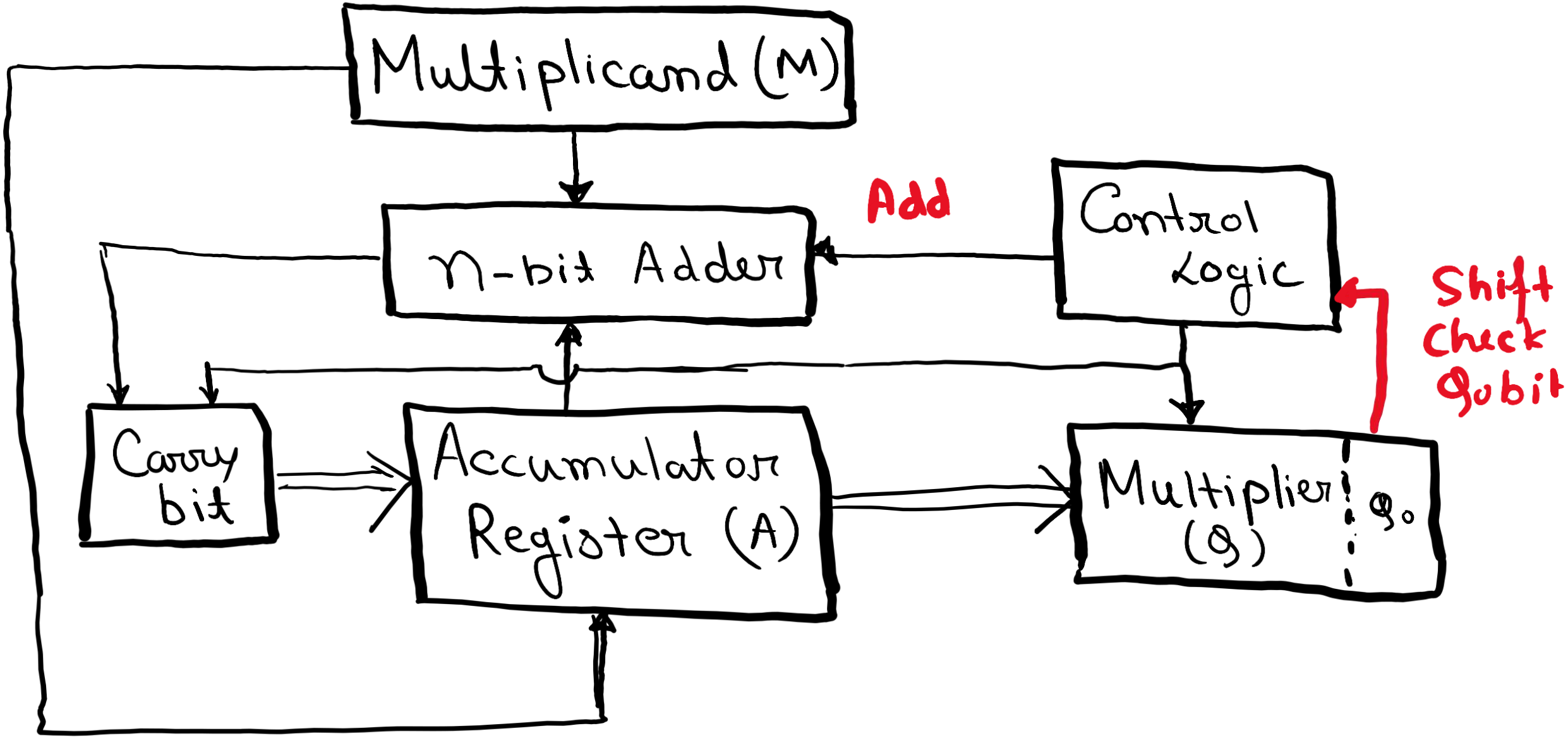
A → 4bit or n-bit (Accumulator)

→ No. of Addition operation = No. of 1 is present in Q.

→ No. of Shift Operation Perform = No. of bits in M & Q

Here Addition = 3 & Shift = 4

Hardware :-


Multiplicand (M)
n-bit Adder
Add
Control Logic
Shift
Check
Qubit
Carry bit
Accumulator Register (A)
Multiplier (Q)
q_0

Booth's Algorithm (Signed Multiplication)

e.g.: $(-7)_{10} \times (+3)_{10} = (-21)_{10}$

$(-7)_{10}$ → Multiplicand (M)

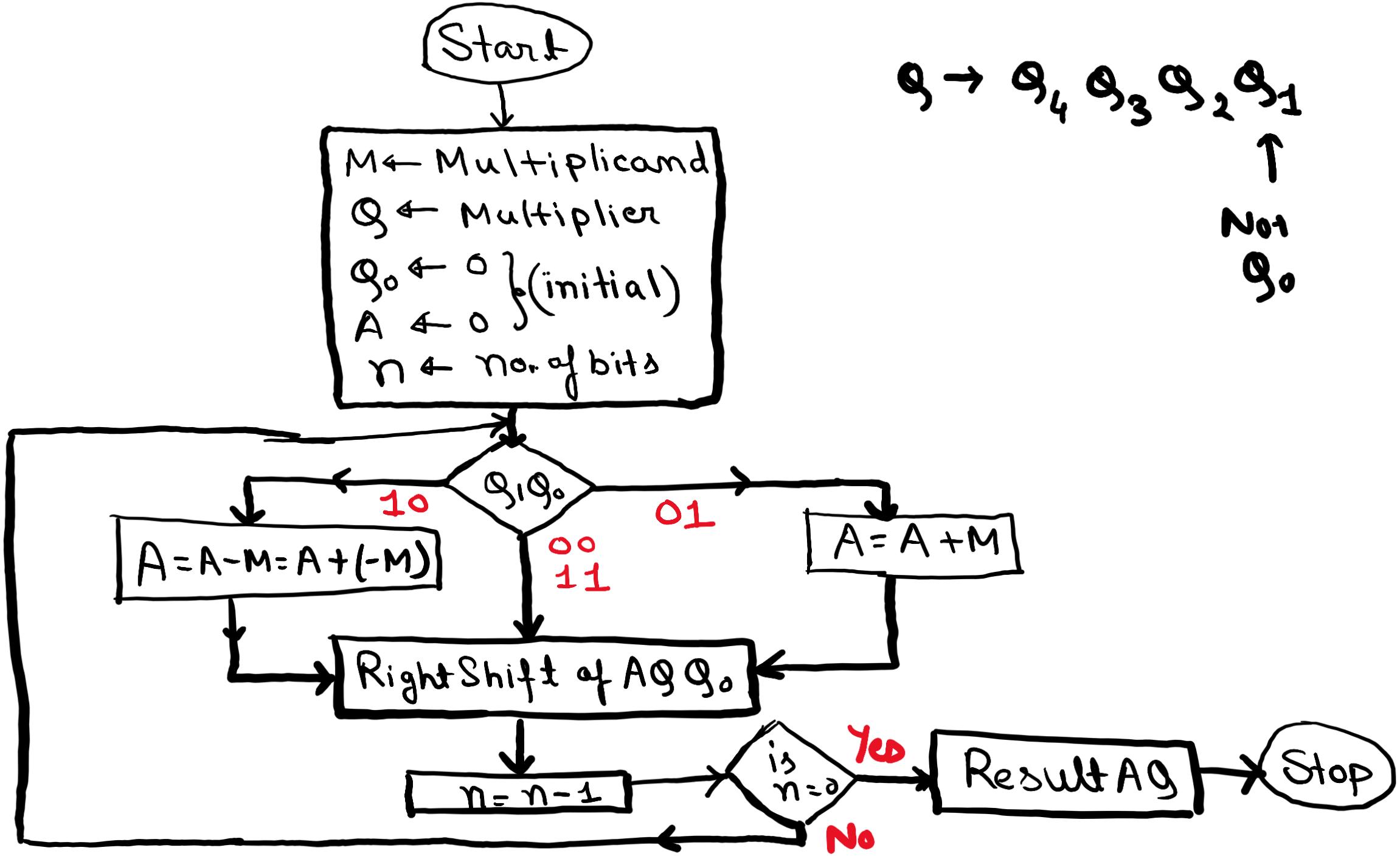
$(+3)_{10}$ → Multiplier (Q)

$(-21)_{10}$ → Product

$(Q_4\ Q_3\ Q_2\ Q_1) = (0\ 0\ 1\ 1)_2$

Now, $M = (-7)_{10}$ ⟶ $-M = (7)_{10} = (0\ 1\ 1\ 1)_2$

$M = (-7)_{10}$ → 2's Complement form → $M = (-7)_{10} = (1\ 0\ 0\ 1)_2$


Start
M ← Multiplicand
Q ← Multiplier
Q0 ← 0
A ← 0
(initial)
n ← no. of bits
Q1Q0
10
01
00
11
A = A - M = A + (-M)
A = A + M
Right Shift of AQQ0
n = n - 1
is n = 0
Yes
No
Result AQ
Stop
Q → Q4 Q3 Q2 Q1
Not
Q0

n	A	Q	Q_0	Operation
4	0 0 0 0	Q_4 Q_3 Q_2 Q_1 0 0 1 1	0	Initialization
	0 1 1 1	0 0 1 1	0	As $(Q_1Q_0) \rightarrow (10)$; A = A − M = A + (−M)
	0 0 1 1	1 0 0 1	1	Right Shift (RS) → AQQ_0
3	0 0 0 1	1 1 0 0	1	As $(Q_1Q_0) \rightarrow (11) \Rightarrow$ RS → AQQ_0
2	1 0 1 0	1 1 0 0	1	As $(Q_1Q_0) \rightarrow (01) \Rightarrow$ A = A + M 0001 + 1001
	1 1 0 1	0 1 1 0	0	RS → AQQ_0
1	1 1 1 0	1 0 1 1	0	As $(Q_1Q_0) \rightarrow (00) \Rightarrow$ RS → AQQ_0
0	1 1 1 0	1 0 1 1	0	Stop

AQ → 1 1 1 0 1 0 1 1

→ if MSB is '1' → it is a -ve number → 2's Complement is required to get final result

→ if MSB is '0' → it is a +ve number → AQ is final result

AQ → 1 1 1 0 1 0 1 1

2^4 2^3 2^2 2^1 2^0

$(0\ 0\ 0\ 1\ \ 0\ 1\ 0\ 1)_2 \rightarrow (-21)_2$

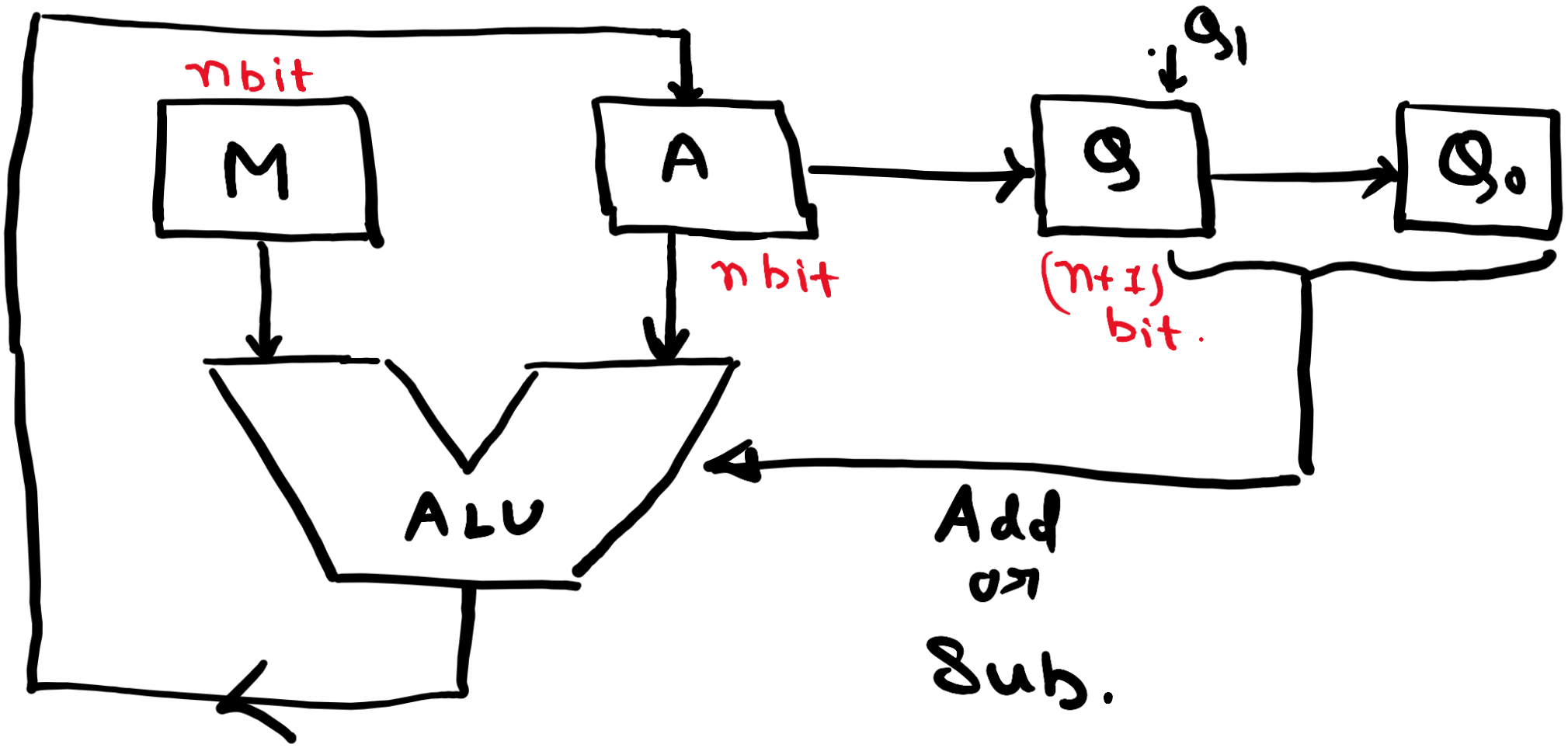
for MSB is 1

Using Booth's Algorith for Multiplication, reduce no. of addition operation Compare to Add & Shift method.

Here, Addition Operation Required = 2

& Shift operation Required = 4

Hardware implementation



Design of Divisor

Slow Algorithms

Iterative Operation : Subtraction

- Restoring Division
- Non-Restoring Division

Fast Algorithms

Iterative Operation : Multiplication

- Division by series expansion
- Newton-Raphson Method

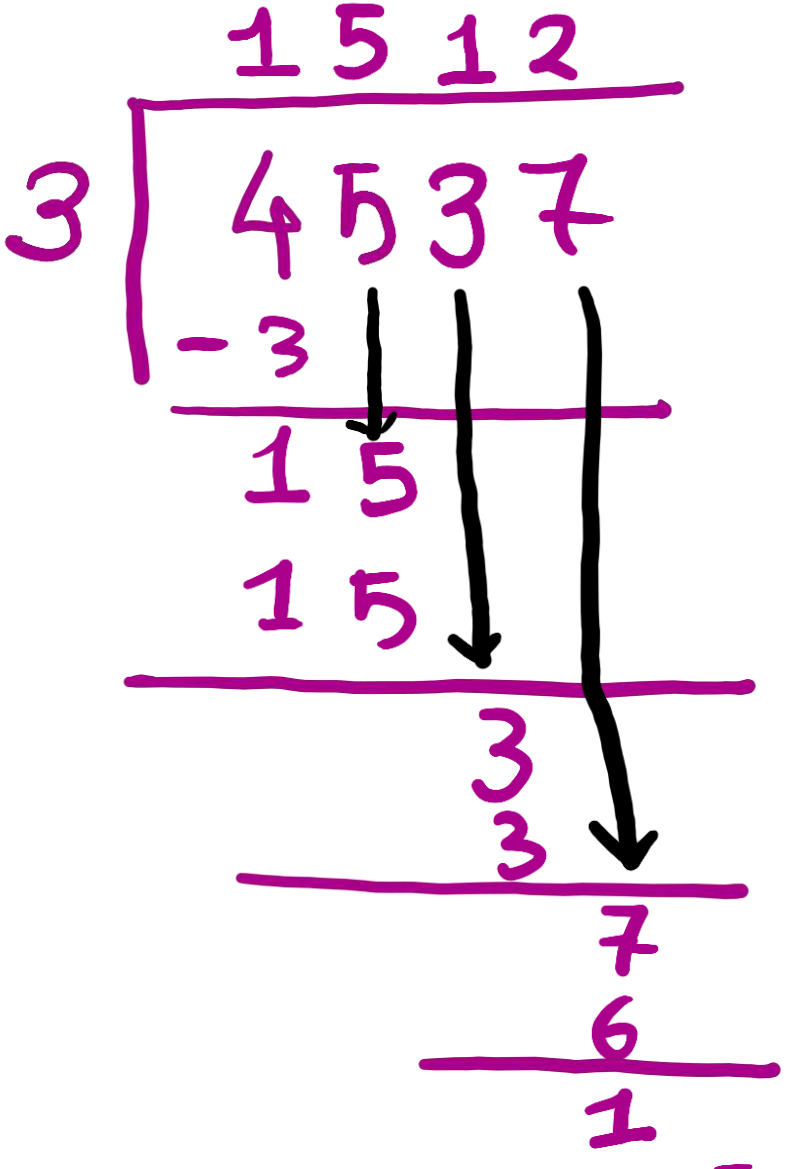
* The Concept of Restoring Division :- (Unsigned no.)

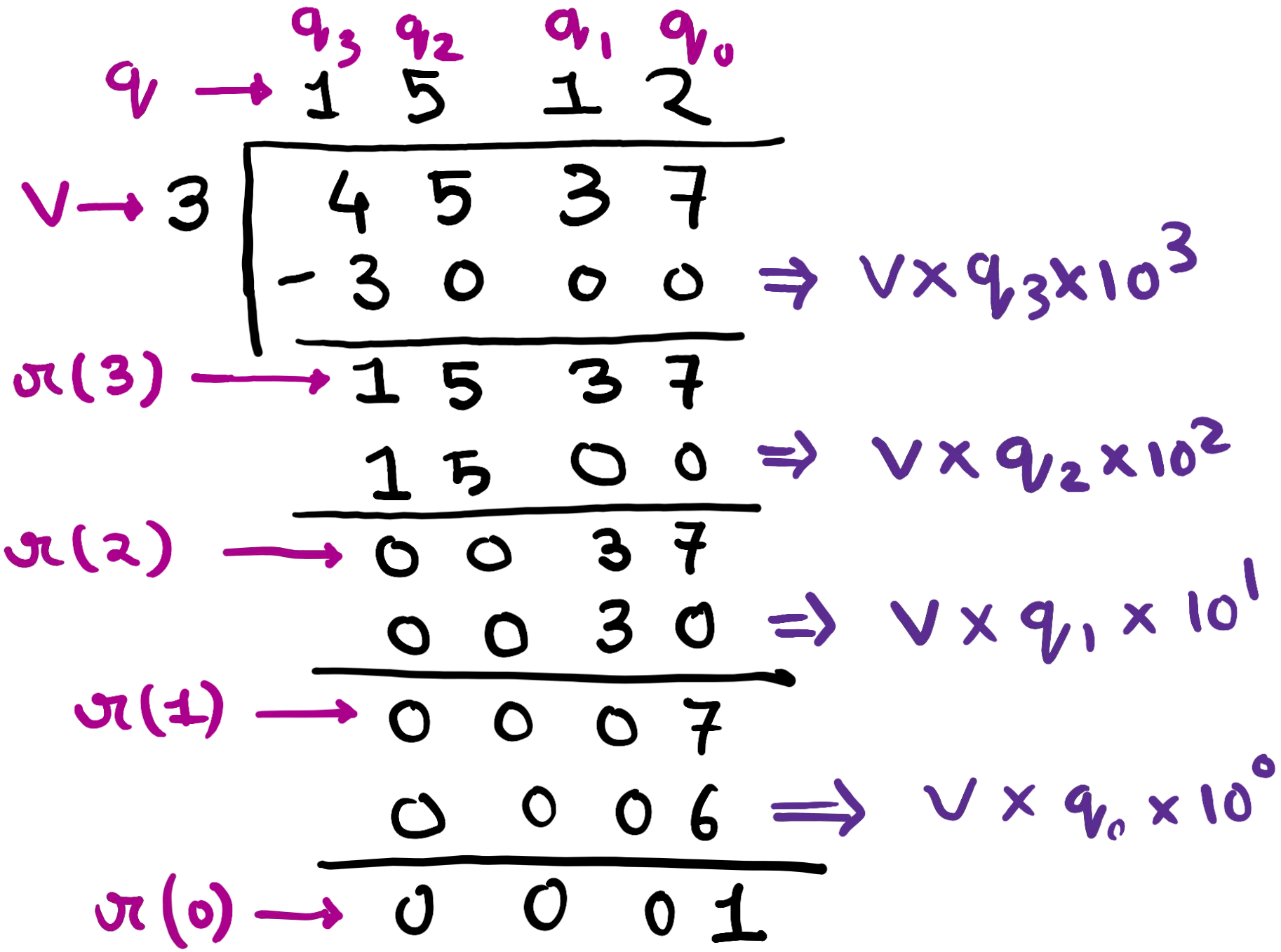
→ Dividend (d) = 4537

→ Divisor (v) : 3

→ Quotient (q) : 1512

→ Remainder (r) : 1




q3 q2 q1 q0
q → 1 5 1 2
V → 3
4 5 3 7
- 3 0 0 0 ⇒ V × q3 × 10^3
r(3) → 1 5 3 7
1 5 0 0 ⇒ V × q2 × 10^2
r(2) → 0 0 3 7
0 0 3 0 ⇒ V × q1 × 10^1
r(1) → 0 0 0 7
0 0 0 6 ⇒ V × q0 × 10^0
r(0) → 0 0 0 1

1st iteration : $4537 - 3 \times 1 \times 10^3 = 1537$

2nd iteration : $1537 - 3 \times 5 \times 10^2 = 0037$

3rd iteration : $0037 - 3 \times 1 \times 10^1 = 0007$

4th iteration : $0007 - 3 \times 2 \times 10^0 = 0001$

∴ Reminder function for i^{th} iteration,

$$r(i) = r(i+1) - q_i \times V \times 10^i$$

→ Restoring division process,

$q_3\ q_2\ q_1\ q_0$

dividend (d) = 4537, Divisor (v) = 3, quotient (q) =

$4537 - 3\times10^3 = 1537;\ q_3 = 1$

$1537 - 3\times10^3 = -1463;\ q_3 = 2$

$-1463 + 3\times10^3 = 1537;\ q_3 = 1$

{ -ve reminder not possible, so restore the value by adding

$1537 - 3\times10^2 = 1237;\ q_2 = 1$

$1237 - 3\times10^2 = 937;\ q_2 = 2$

$937 - 3\times10^2 = 637;\ q_2 = 3$

$\Rightarrow$ { q_i also implies no. of subtraction is performed

$637 - 3 \times 10^2 = 337, \; q_2 = 4$

$337 - 3 \times 10^2 = 37, \; q_2 = 5$

-ve reminder is not possible. so restore the value

$37 - 3 \times 10^2 = -263, \; q_2 = 6$

$-263 \oplus 3 \times 10^2 = 37$, $q_2 = 5$ → Restoring

$37 - 3 \times 10^1 = 7, \; q_1 = 1$

$7 - 3 \times 10^1 = -23, \; q_1 = 2$

$-23 \oplus 3 \times 10^1 = 7$, $q_1 = 1$ → Restoring

$7 - 3 \times 10^0 = 4$, $q_0 = 1$

$4 - 3 \times 10^0 = 1$, $q_0 = 2$

$1 - 3 \times 10^0 = -2$, $q_0 = 3$

$-2 \oplus 3 \times 10^0 = 1$, $q_0 = 2$ Restoring

Final remainder.

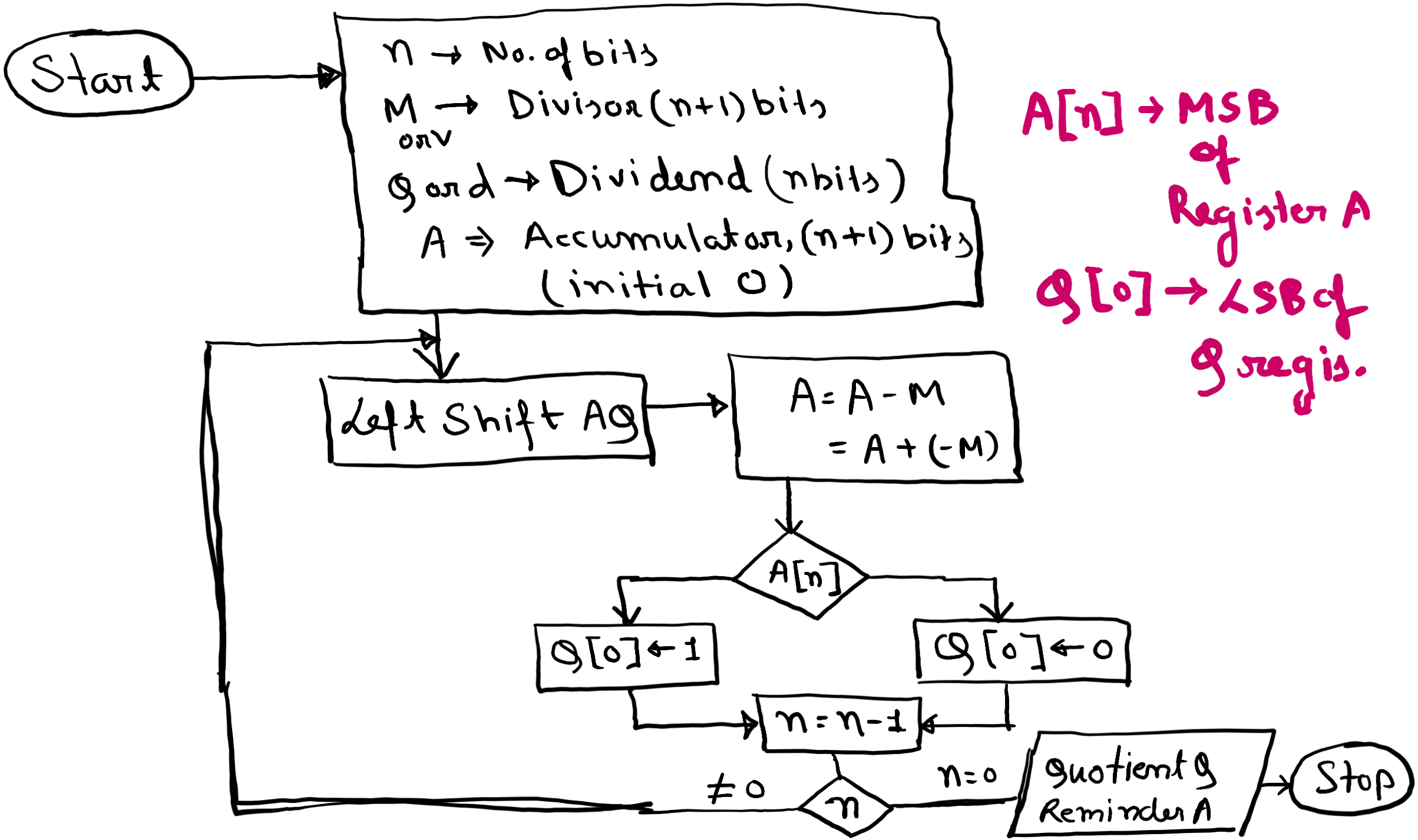
$d = 4537$

$v = 3$

$r = 1$

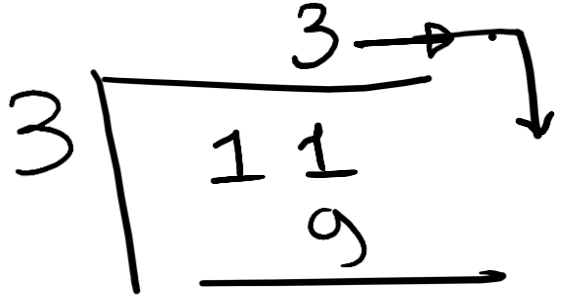
$q_3\ q_2\ q_1\ q_0$

$Q = 1512$


Start
n → No. of bits
M → Divisor (n+1) bits
or V
Q or d → Dividend (n bits)
A ⇒ Accumulator, (n+1) bits
(initial 0)
Left Shift AQ
A = A - M
= A + (-M)
A[n]
Q[0] ← 1
Q[0] ← 0
n = n - 1
n
≠ 0
n = 0
Quotient Q
Reminder A
Stop
A[n] → MSB of Register A
Q[0] → LSB of Q regis.

Example :- Dividend (Q or A) = $(11)_{10}$

Divisor (M or V) = $(3)_{10}$



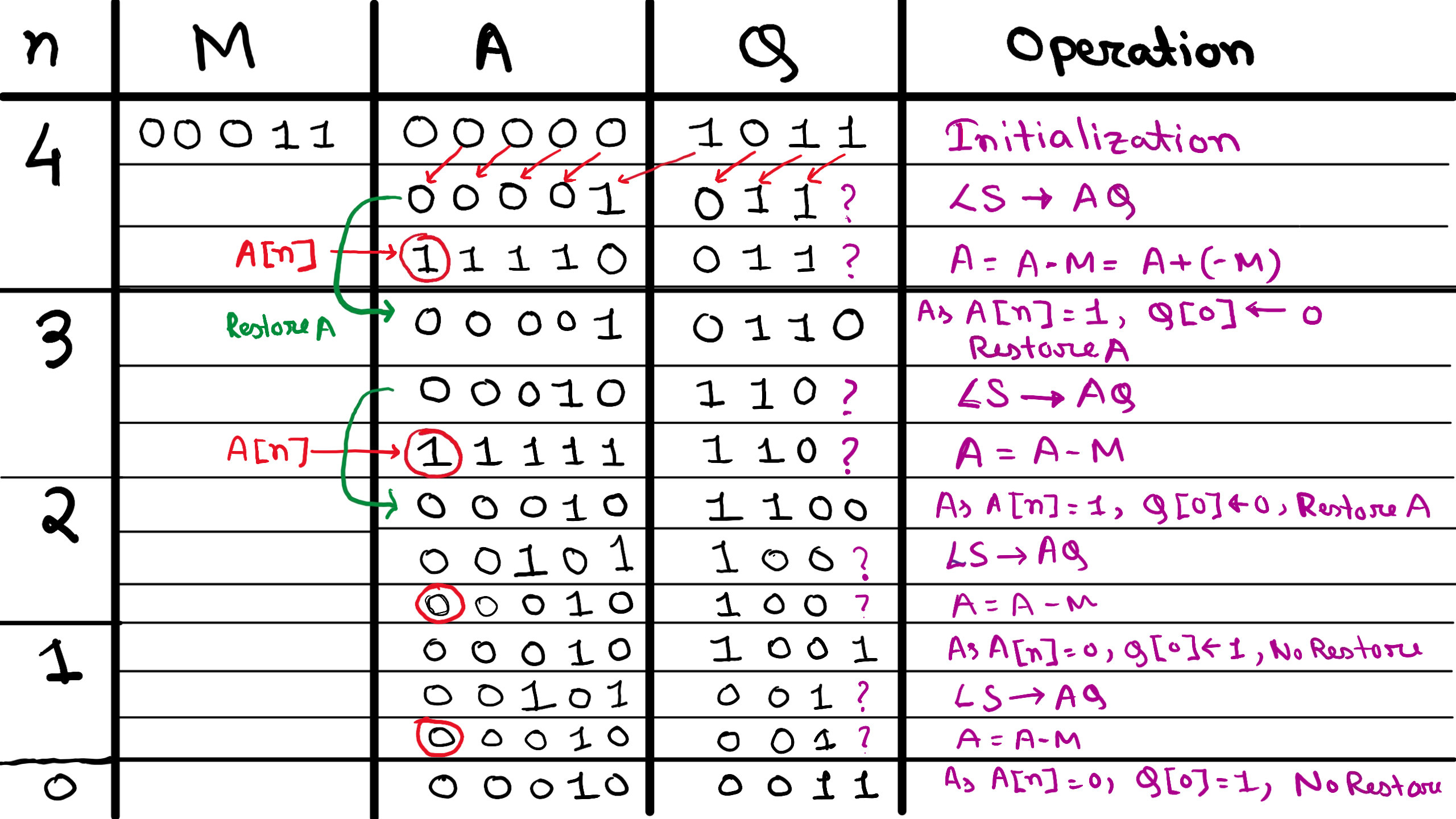
Quotient (Result stored at Q)

Reminder (Result stored at A)

$M \rightarrow (3)_{10} \rightarrow (11)_2 \rightarrow (00011) \Rightarrow (n+1) \Rightarrow (4+1)$ bit

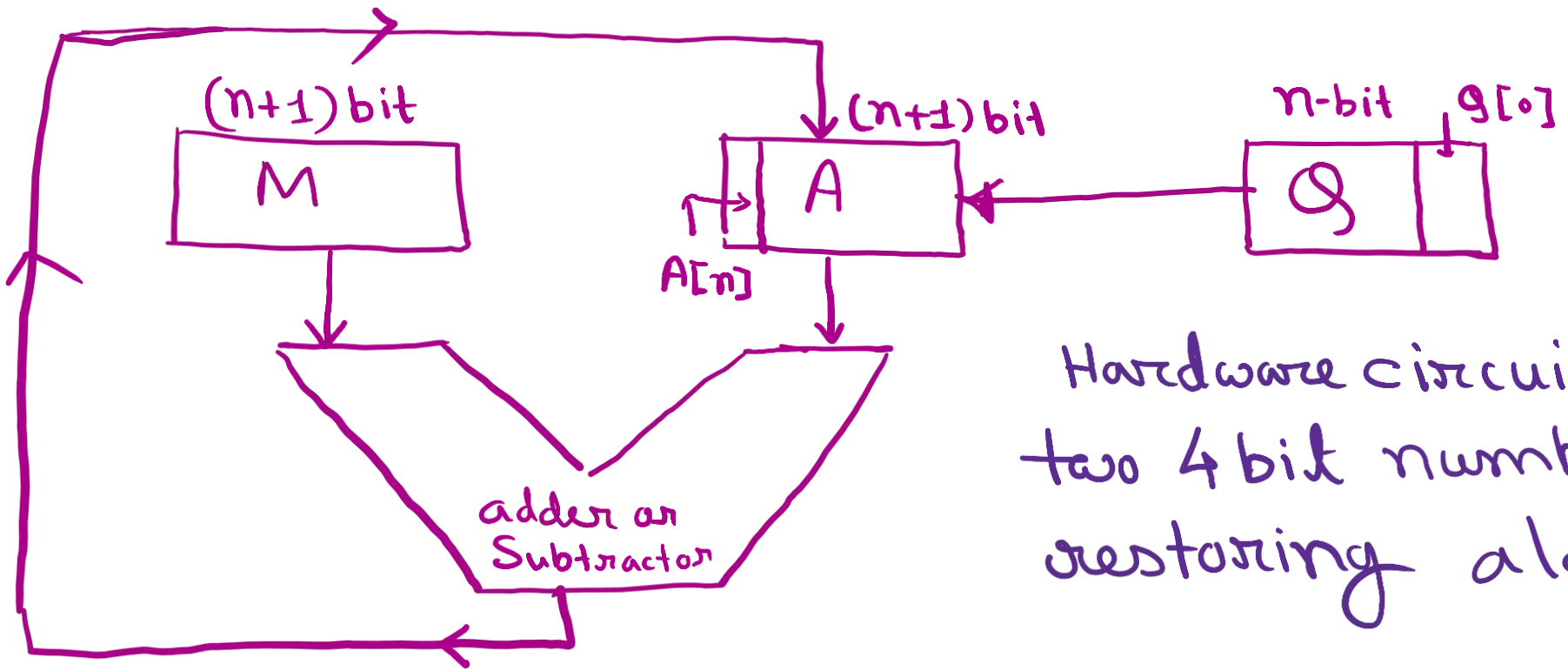
$Q \rightarrow (11)_{10} \rightarrow (1011)_2 \Rightarrow$ 4 bit

$-M \rightarrow$ 2's comp of $M \rightarrow (11101)_2 \Rightarrow$ 5 bit.


n
M
A
Q
Operation
4
00011
00000
1011
Initialization
00001
0111?
ASQ
A[n]
11110
0110?
A = A-M = A+(-M)
3
Restore A
00001
0110
As A[n]=1, Q[0]←0
Restore A
00010
110?
ASQ
A[n]
11111
110?
A = A-M
2
00010
1100
As A[n]=1, Q[0]←0, Restore A
00101
100?
LSAQ
00010
100?
A = A-M
1
00010
1001
As A[n]=0, Q[0]←1, No Restore
00101
001?
LSAQ
00010
001?
A = A-M
0
00010
0011
As A[n]=0, Q[0]=1, No Restore

$\therefore$ Reminder, $r = [A] = (00010)_2 = (2)_{10}$ ..(A)

$\therefore$ Quotient, $q = (0011)_2 = (3)_{10}$ ..(A)



Hardware circuit to divide two 4 bit numbers using restoring algorithm

* Non-Restoring Division Algorithm of Unsigned Number

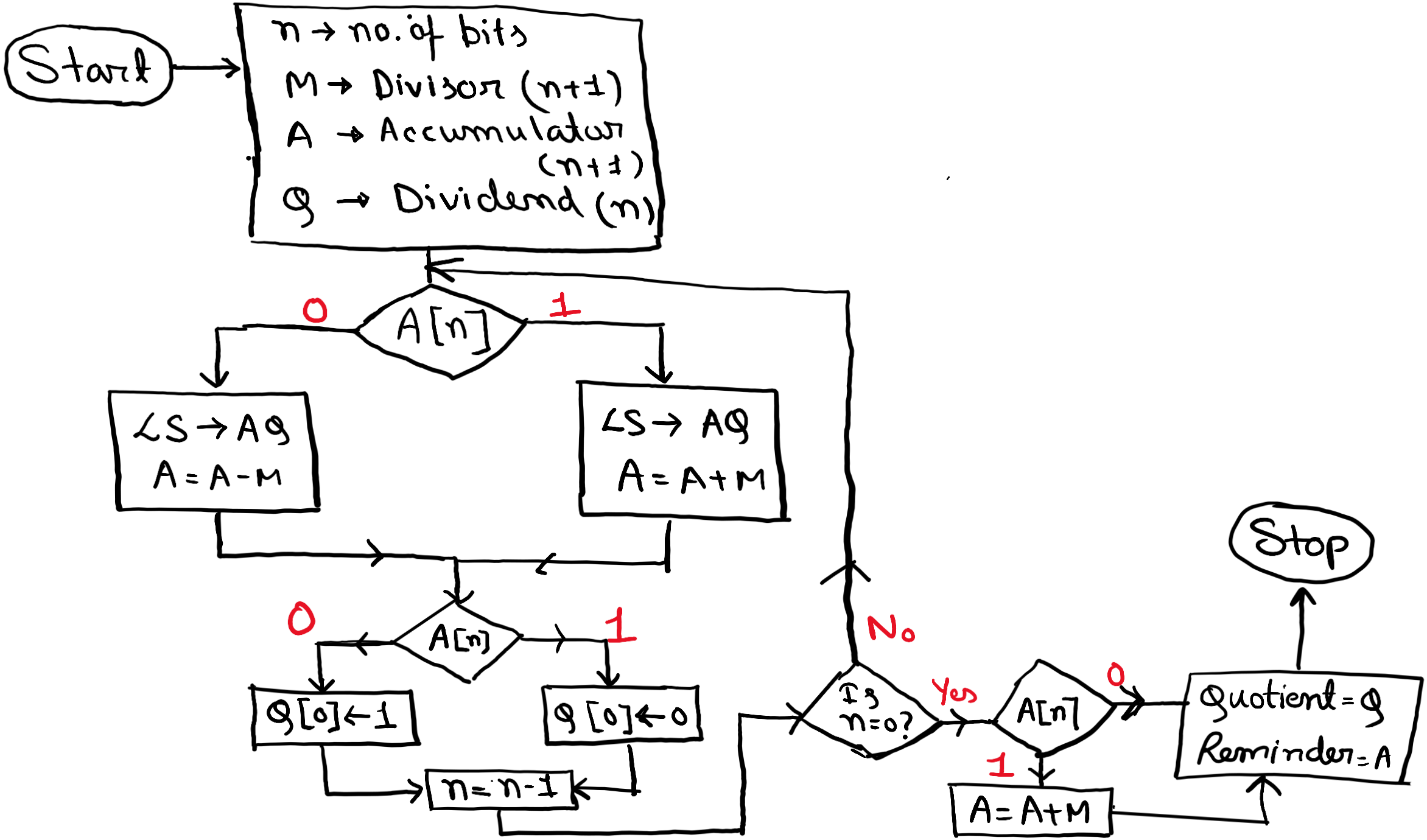
→ Dividend (Q) = $(11)_{10} = (1011)_2$ → n bit

→ Divisor (M) = $(3)_{10} = (00011)_2$ → (n+1) bit

→ Quotient (q) = $(3)_{10} = (0011)_2$ → n-bit (Q)

→ Reminder (r) = $(2)_{10} = (00010)_2$ → (n+1) bit (A)

→ −M = 2's Compliment form of M = $(11101)_2$


Start
n → no. of bits
M → Divisor (n+1)
A → Accumulator (n+1)
Q → Dividend (n)
A[n]
0
1
LS → AQ
A = A - M
LS → AQ
A = A + M
A[n]
0
1
Q[0] ← 1
Q[0] ← 0
n = n - 1
Is n=0?
No
Yes
A[n]
0
1
A = A + M
Quotient = Q
Reminder = A
Stop

n	M	A	Q	Operation
4	00011	00000	1011	Initialization
		00001	011?	A Q <<
	A[n] →	11110	011?	A[n]=0 → A=A-M=A+(-M)
.		11110	0110	A[n]=1 ; Q[0]=0
3		11100	110?	A Q <<
		11111	110?	A[n]=1 ; A=A+M
		11111	1100	A[n]=1 ; Q[0]=0
2		11111	100?	A Q <<
		00010	100?	A[n]=1 ; A=A+M
		00010	1001	A[n]=0 ; Q[0]←1
1		00101	001?	A Q <<
		00010	001?	A[n]=0 ; A=A-M
		00010	0011	A[n]=0 ; Q[0]←1

$[A] = (00010)_2 = (2)_{10} \Rightarrow$ Reminder

A[n] = 0

$[Q] = (0011)_2 = (3)_{10} \Rightarrow$ Quotient.