Computer Organization & Architecture

Unit 2

* Arithmetic and Logic Unit

→ performs arithmetic and logical operations on data

→ data is stored in registers and the results of an operation are also stored in registers

→ The CU controls the operation of the ALU & the movement of data in and out of the ALU.

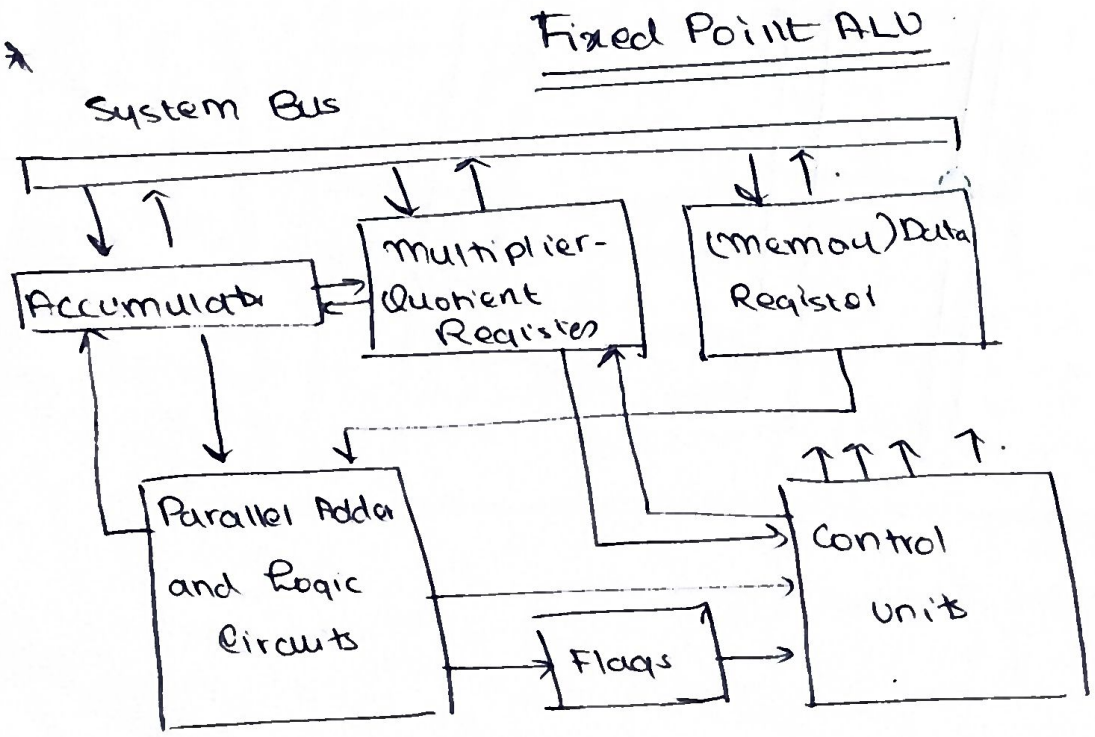
→ Most computers have registers called the Accumulator (AC) or general purpose registers.

→ The AC is the basic register containing one of the operands during operation in the ALU.

→ If the operation is add, the number stored in the AC is the augend and the addend will be located and these operands (addend and augend) are added up & the result is stored back in the AC.

→ The original augend will be lost in AC after addition.

* Fixed Point ALU



→ There are 3 one word registers used for operand storage

(i) Accumulator AC

(ii) Multiplier Quotient Register MQ

(iii) Data Register DR

(iv) Parallel adder that receives inputs from the AC & DR and places its results in the AC.

MQ stores multiplier / quotient

DR stores the multiplicand / divisor.

The results of multiplication & division are stored in the AC/MQ.

Operations

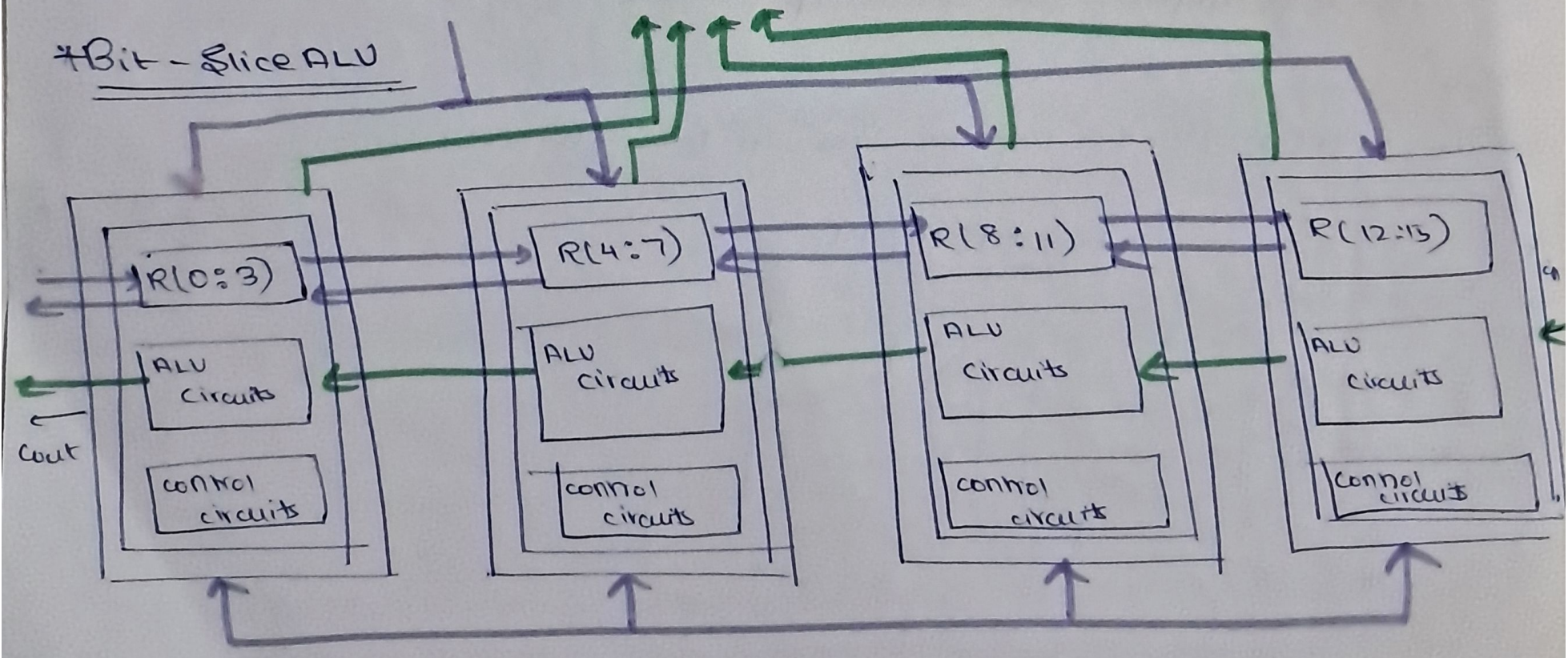
Addition: AC ← AC + DR Multiplication

Subtract: AC ← AC − DR

Multiplication: AC.MQ ← DR * MQ

Division AC.MQ ← MQ / DR

*Bit-Slice ALU



→ Bit-slicing : a technique for constructing a processor from modules of processors of smaller bit width, for the purpose of increasing the word length.

→ The control lines select and sequence the operation to be performed are connected to each slice.

→ Each slice performs the same operation on a diff 4 bit part of the input operands, and produces only the corresponding part of the results

* Integer Representation- Fixed Point

A. Signed Magnitude Representation

→ +ve and -ve numbers are differentiated by the MSB, as a sign bit.

→ If the sign bit is 0, the no is +ve

→ If the sign bit is 1, the no is -ve

eg. +18 = 0 0010010

↓ ↓

one sign bit 7 bit magnitude

Drawbacks

1. Addition & subtraction requires both the sign bit & the magnitude bits to be considered.

2. There are 2 representations for 0

+0 → 0 0000000

+1 → 1 0000000

B. Two's Complement Representation

→ -ve nos represented by their 2's complement

eg. -4

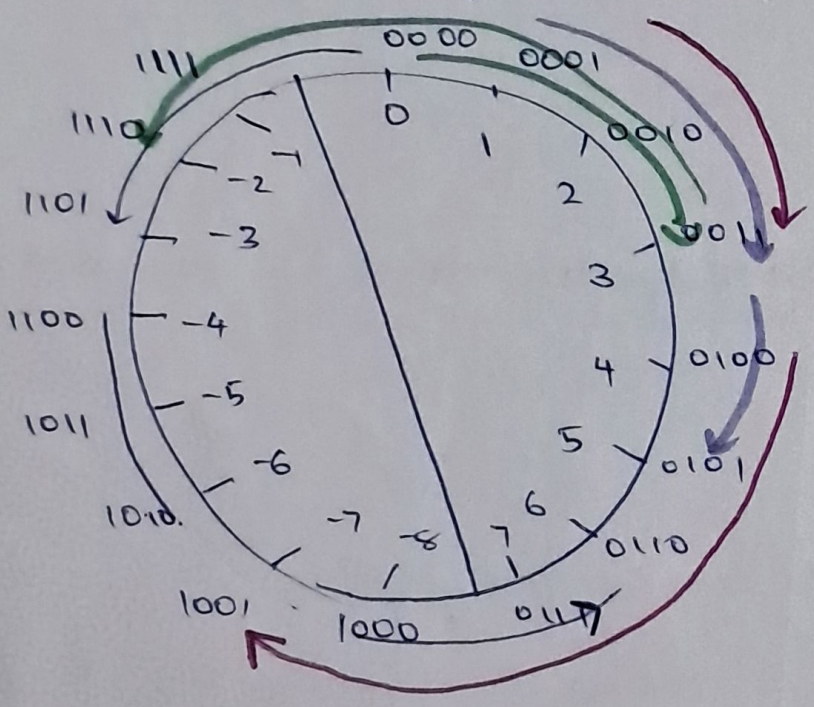
Signed magnitude : 1100

1s complement : 1011

2s complement : 1100

Advantages → only one arithmetic operation is required while subtracting using 2's complement notation

Consider a 4 bit representation:



3+2=5

3+(-5)= -2

3+6=-7 (overflow)

-3+(-6) =-7 (underflow)

Detecting overflow : (i) when adding 2 +ves yields a -ve

(ii) adding 2 -ves yields a +ve

* Sign Extension

In signed magnitude : fill all bits left of the sign bit with 0.

In 2's complement form: ~~move the sign~~ fill the remaining bits with copies of the sign bit.

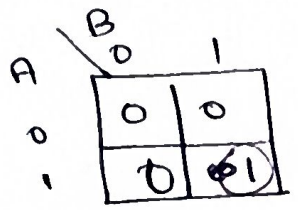
* Half-Adder

→ used to add 2 numbers each of 1 bit

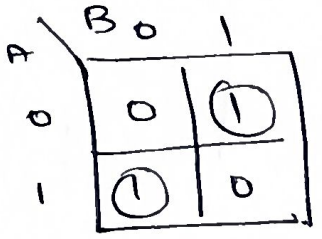
→ Truth Table has sum & carry

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

AB



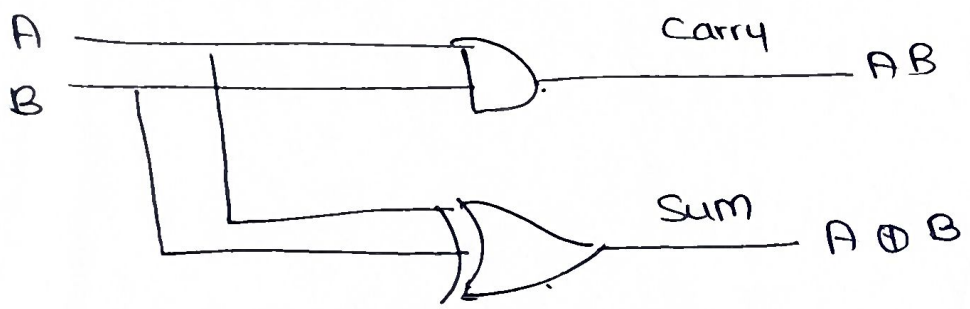
$C = A \cdot B$



$S = AB' + A'B$

$S = A \oplus B$

Circuit Diagram



* Full Adder

→ a combinatorial circuit that performs the sum of 3 input bits

→ has 3 inputs & 2 outputs

→ There are 2 input bits x & y (which are the 2 significant bits to be added)

→ The third bit z, represents the carry from the previous lower significant position.

Truth Table

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Sum

x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	1	0

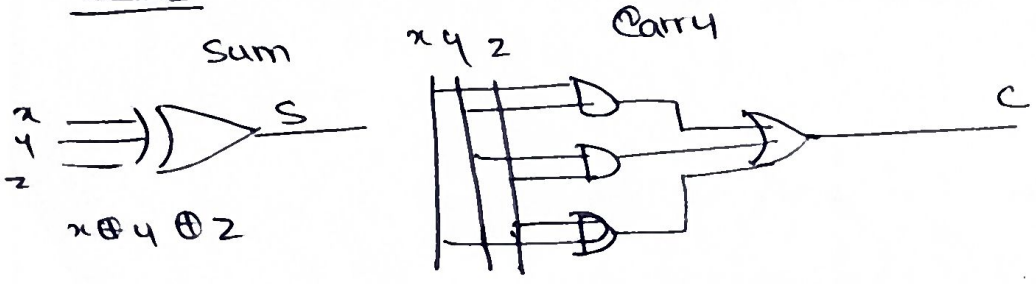
Sum = $x \oplus y \oplus z$

Carry

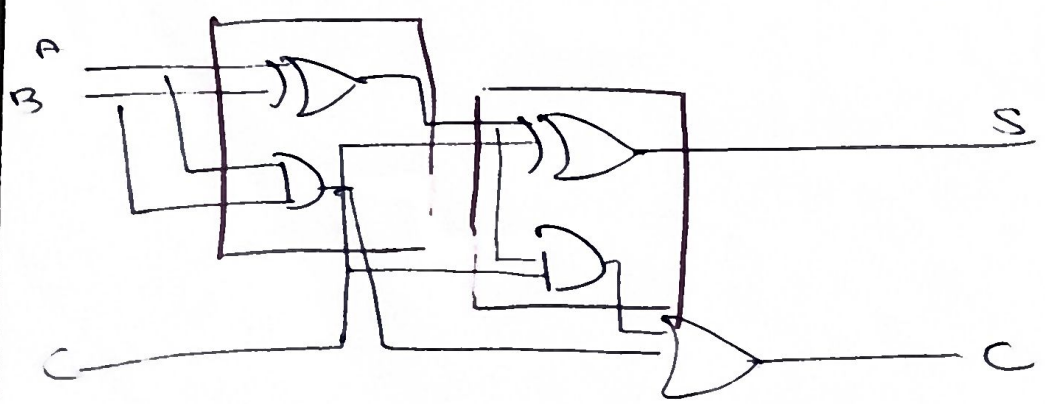
x \ yz	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Carry = $xy + xz + yz$

Circuit



From 2 halfadders



* 4 bit binary adder

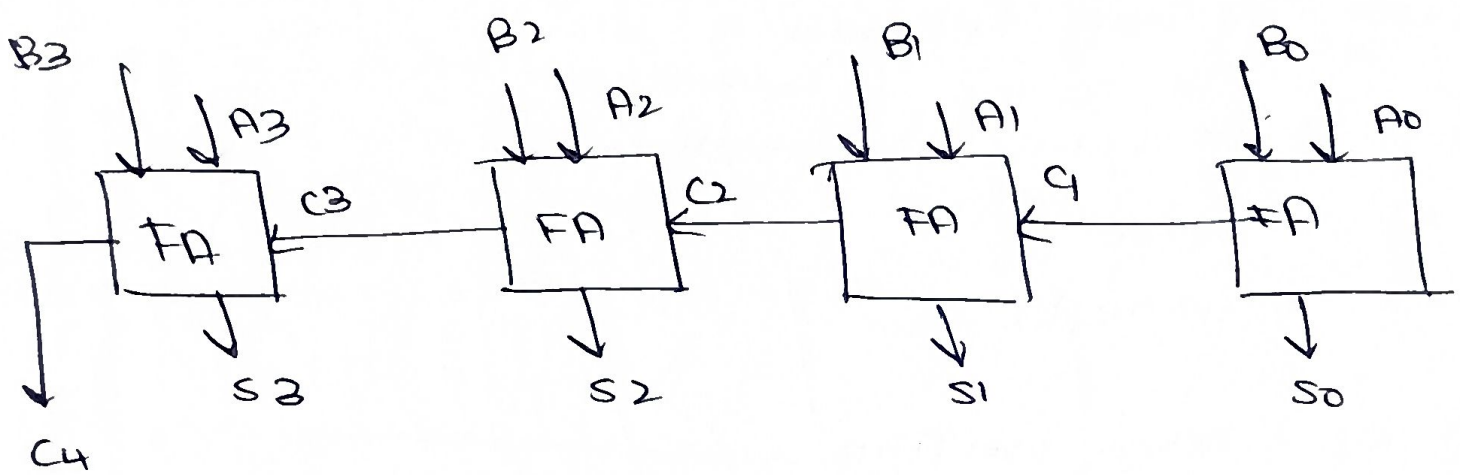
→ produces the arithmetic sum of 2 binary numbers

→ made using full adders connected in cascade w/ the op carry from each full adder connected to the i/p carry of the next fulladder.

→ Addition of n bits requires n full adders

→ Bits are added from the LSB to form the sum bit & the carry bit

→ The input carry C_0 is 0.



Consider the addition of 2nos

A = 1011
B = 0011

This needs 4 Fulladders

Input Carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
output carry	0	0	1	1	C_{i+1}

* 4-bit binary adder / subtractor

→ Subtraction is done by taking complements

→ A - B is done by taking the 2's complement of B and adding it to A.

→ The circuit for subtracting A - B has an adder, along w/ inverters between each data input for B & the full adder input.

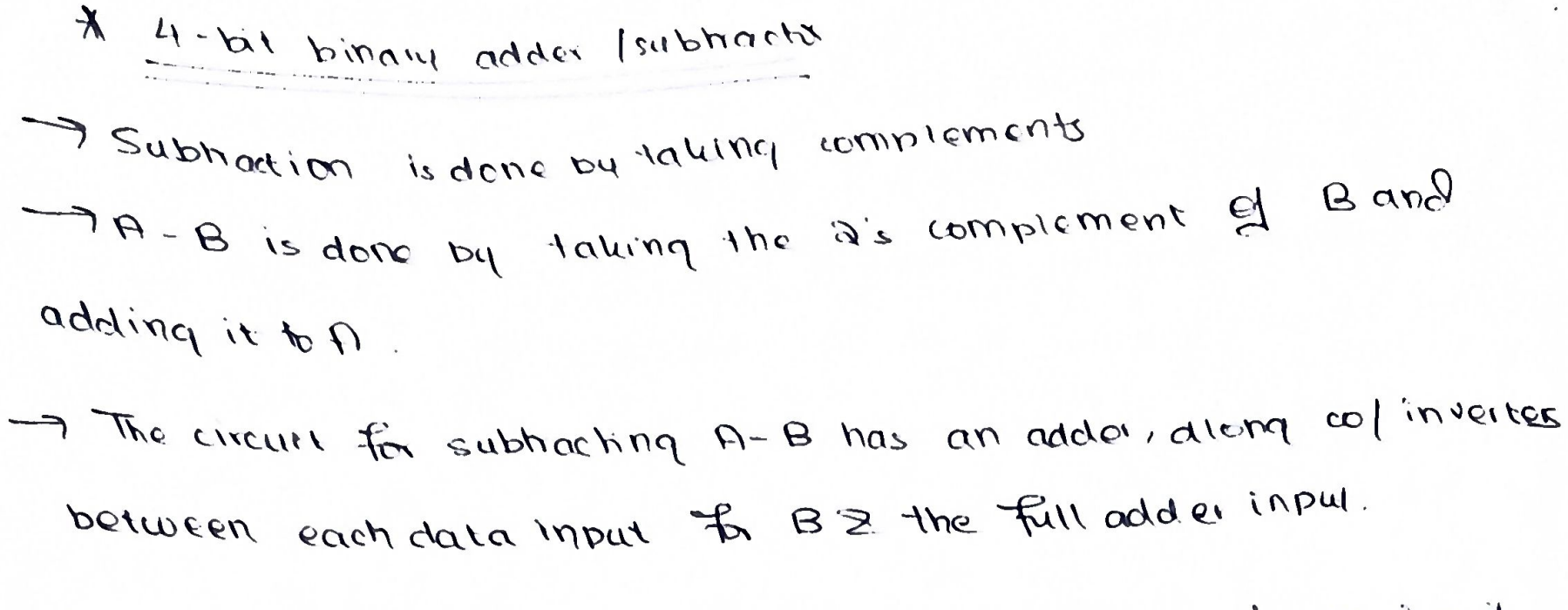
→ Addition and subtraction can be combined into a single circuit by an ex-OR w/ each full adder.

→ The mode input M controls the operations

M = 0 ⇒ adder

M = 1 ⇒ subtractor

→ The ex-OR w/ V detects overflow.



Overflow → 2 nos w/ n digits each are added and the sum is a no. occupying n+1 digits, then we say that an overflow has occured

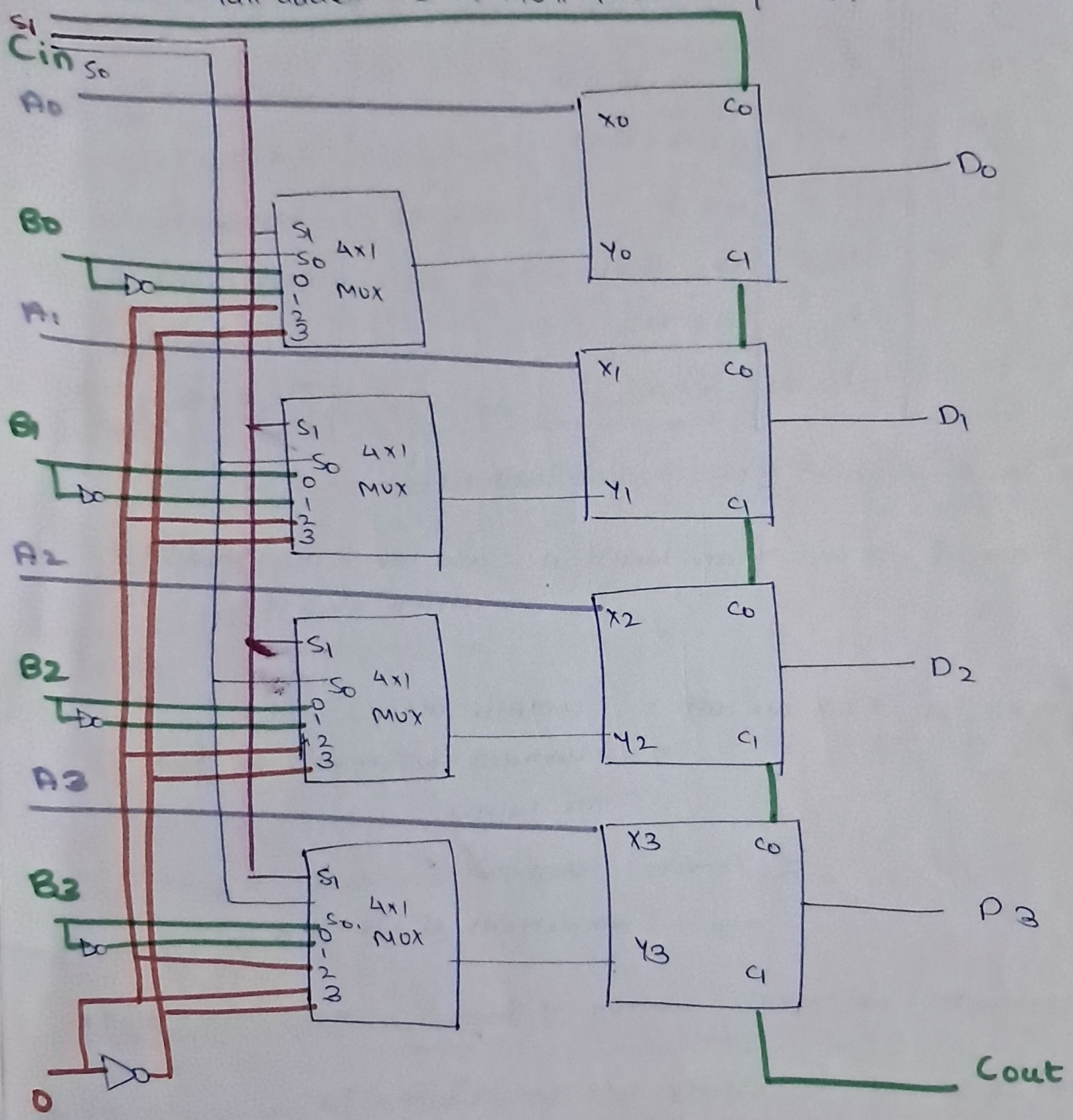
V detects overflows

C3	C4	V = C3 ⊕ C4	
0	0	0	
0	1	1	overflow
1	0	1	overflow
1	1	0	

* 4 bit arithmetic circuit

→ used to carry out a variety of arithmetic operations

→ has 4 full adders & 4 mux for choosing different options



→ mux selects one of the options

→ Ai connected directly to fulladder

→ Bi and $\overline{Bi}$ connected to 0 & 1

→ 0 & 1 connected to input lines 2 & 3.

Selection			In	Output	Operations
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	
0	0	0	B	$D = A + B + 0 = A + B$	addition
0	0	1	B	$D = A + B + 1$	addition w/ carry
0	1	0	$\overline{B}$	$D = A + \overline{B} + 0$	subtraction w/ borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

* Hardware Implementation of Addition & Subtraction

When the signs of X and Y are identical : add the 2 magnitudes
attach sign of X

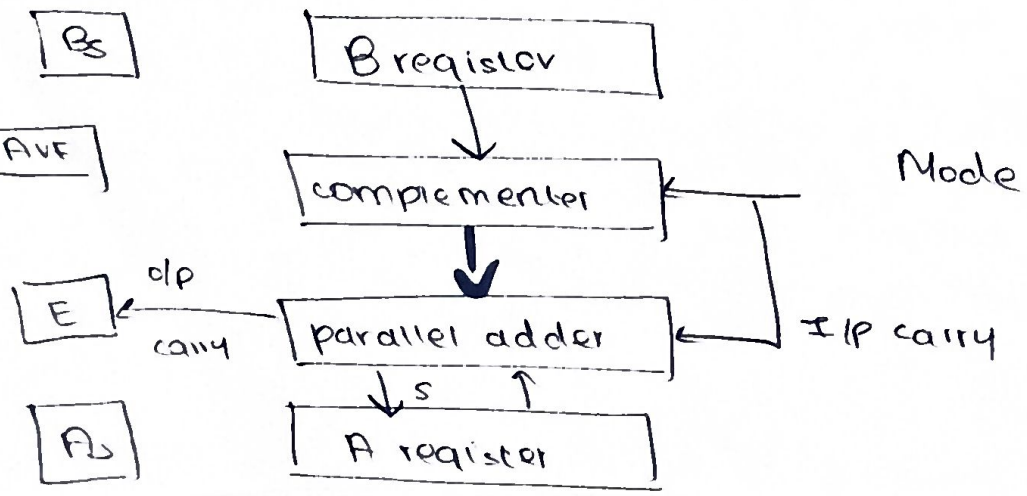
When the signs of X & Y are diff : compare mag.
subtract the smaller no. from
the larger

If $X > Y$ sign = X

else complement & sign of X

If the magnitudes are equal, subtract Y from X, make result positive,

Hardware for signed magnitude addition & subtraction.



→ A and B register store the 2 numbers

→ A_s and B_s store the sign bit of those 2 numbers

→ M = mode input

When M = 0

⇒ I/p carry = 0 ⇒ addition

⇒ complementor will not work

⇒ B will pass through complementer, unaffected and go to parallel adder

⇒ A also enters parallel adder, results in A+B

⇒ If there is a carry, it goes to E. When there is an overflow, it goes to AVF.

When M = 1

⇒ I/p carry = 1 ⇒ subtraction

⇒ B goes to complementer, comes out as $\bar{B}$, then goes to parallel adder

⇒ A goes to parallel adder.

The net result is $A + \bar{B} + 1 = A - B$

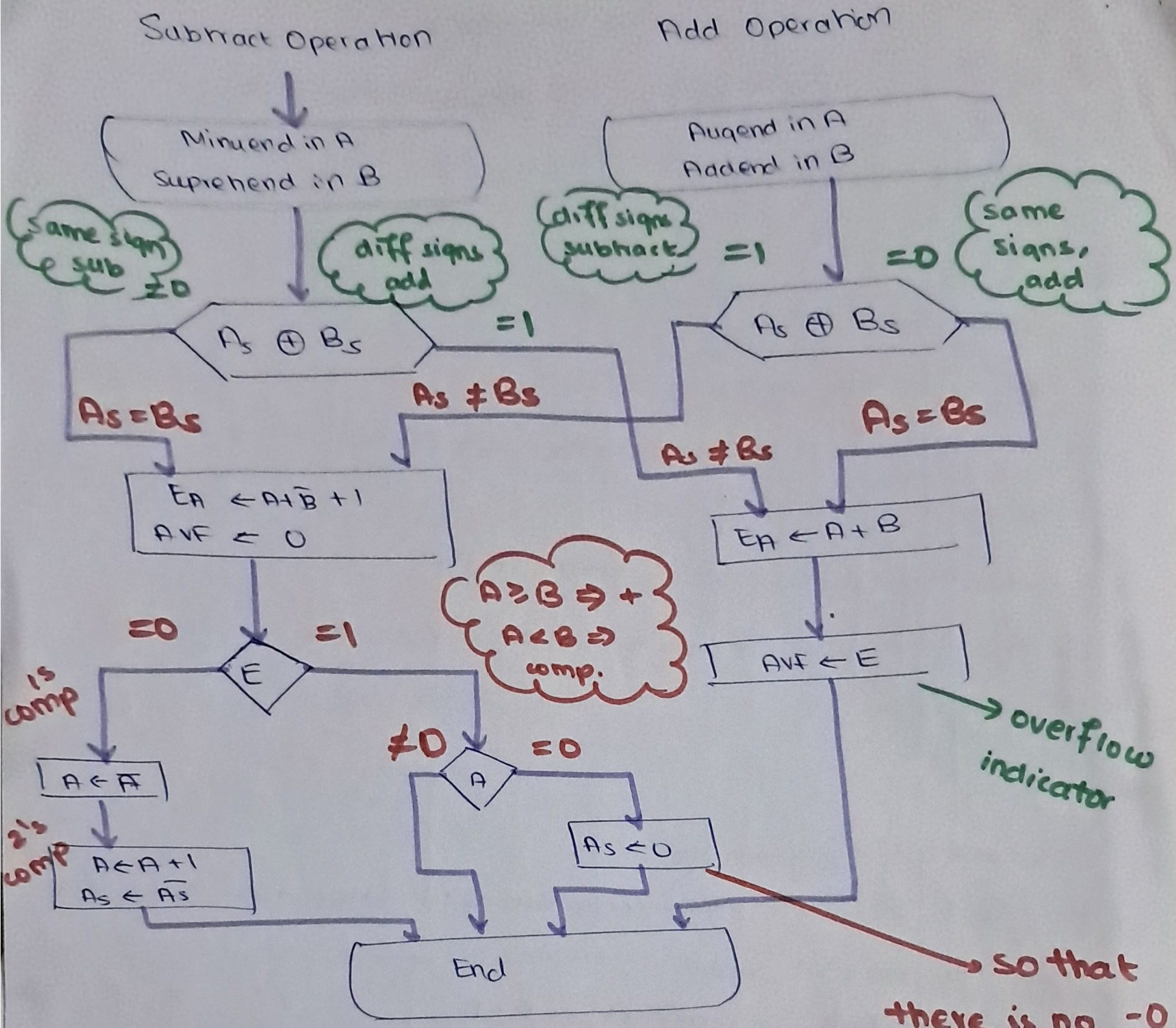
⇒ result in E.

* Flowchart Representation

→ If 2 operands have the same sign, the intended operation is performed

→ If they have diff. signs, then subtraction ⇒ addition is performed

addition ⇒ subtraction is performed



* Signed 2's complement data

Implementation

→ The left most bit of a binary no. represents the sign bit

0 → +ve

1 → -ve

Addition - add the digits along w/ the sign bits

- carry is discarded

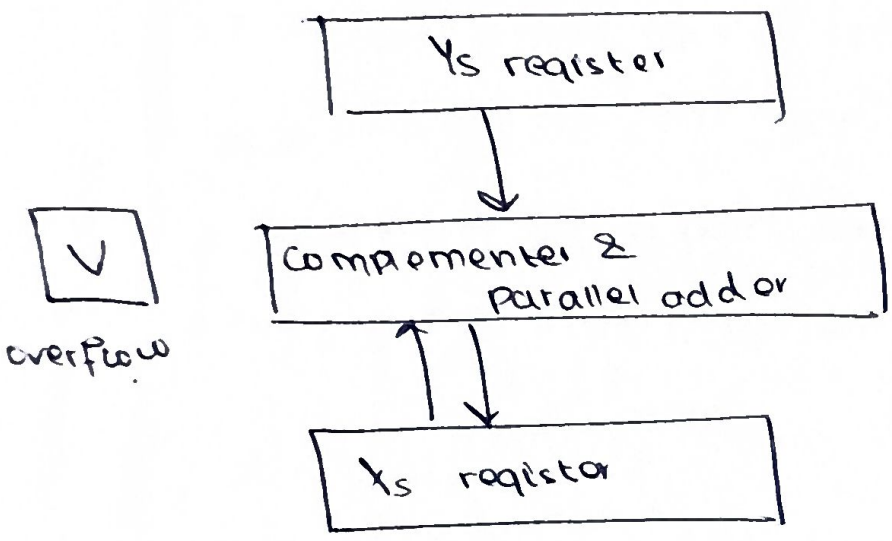
Subtraction

→ take 2's complement of the subtrahend and adding it to the minuend

Overflow : inspect the last 2 carries of the addition

When the 2 carries are applied to the XOR gate, there is an overflow when the o/p is 1.

* Hardware Representation



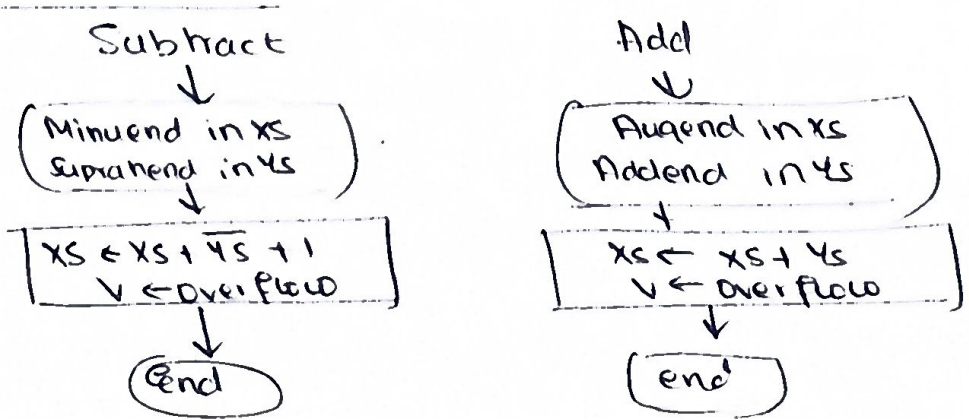
→ The are 2 registers, Xs and Ys, where the left most bits represent the sign bits of the number.

→ The 2 sign bits are added or subtracted together w/ the other bits in the complementer & parallel adder.

→ The overflow flip-flop V is set to 1, if there is an overflow.

→ The output carry is discarded.

* Hardware Algorithm



Example :

Addition

+74 : 0 1 0 0 1 0 10

+69 : 0 1 0 0 0 1 0 1

1 0 0 0 1 1 1 1

Subtraction

125 0 1 1 1 1 1 0 1

-90 1 0 1 0 0 1 1 0 → 2's comp

1 0 0 1 0 0 1 1 = 35

↓ discard

* Carry Propagation

→ Addition of 2 binary nos in parallel would mean that all the bits of the augend and addend are available at the same type.

→ Parallel adders are ripple carry types, they carry the o/p of each full stage adder and connect it to the carry input of the next higher order stage.

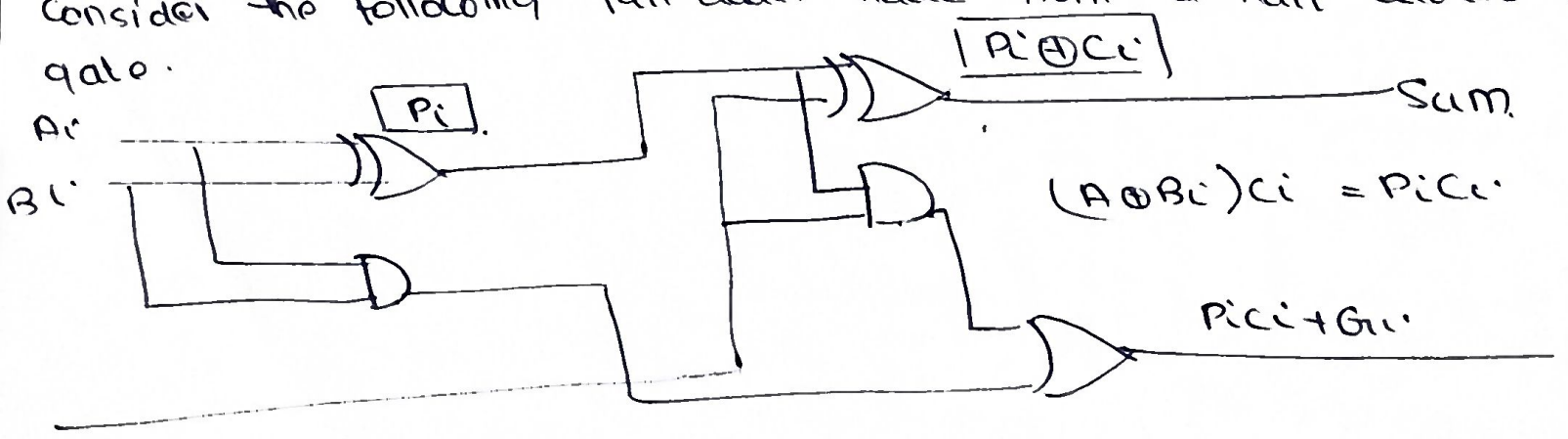
→ The sum of output of any stage cannot be produced until the input carry occurs.

→ This leads to a time delay.

Carry Propagation diagram = 4 bit full adder

* Carry lookahead Adder

Consider the following full adder made from 2 half adders and an OR gate.



$P_i = A_i \oplus B_i$ = carry propagate, determines whether a carry into stage i propagates into stage $i+1$

$G_i = A_i B_i$ = carry generate

$S = P_i \oplus C_i$

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

If C_0 is the input carry

$C_1 = P_0 C_0 + G_0$

$C_2 = P_1 C_1 + G_1$

$= P_1 (P_0 C_0 + G_0) + G_1$

$= P_1 P_0 C_0 + P_1 G_0 + G_1$

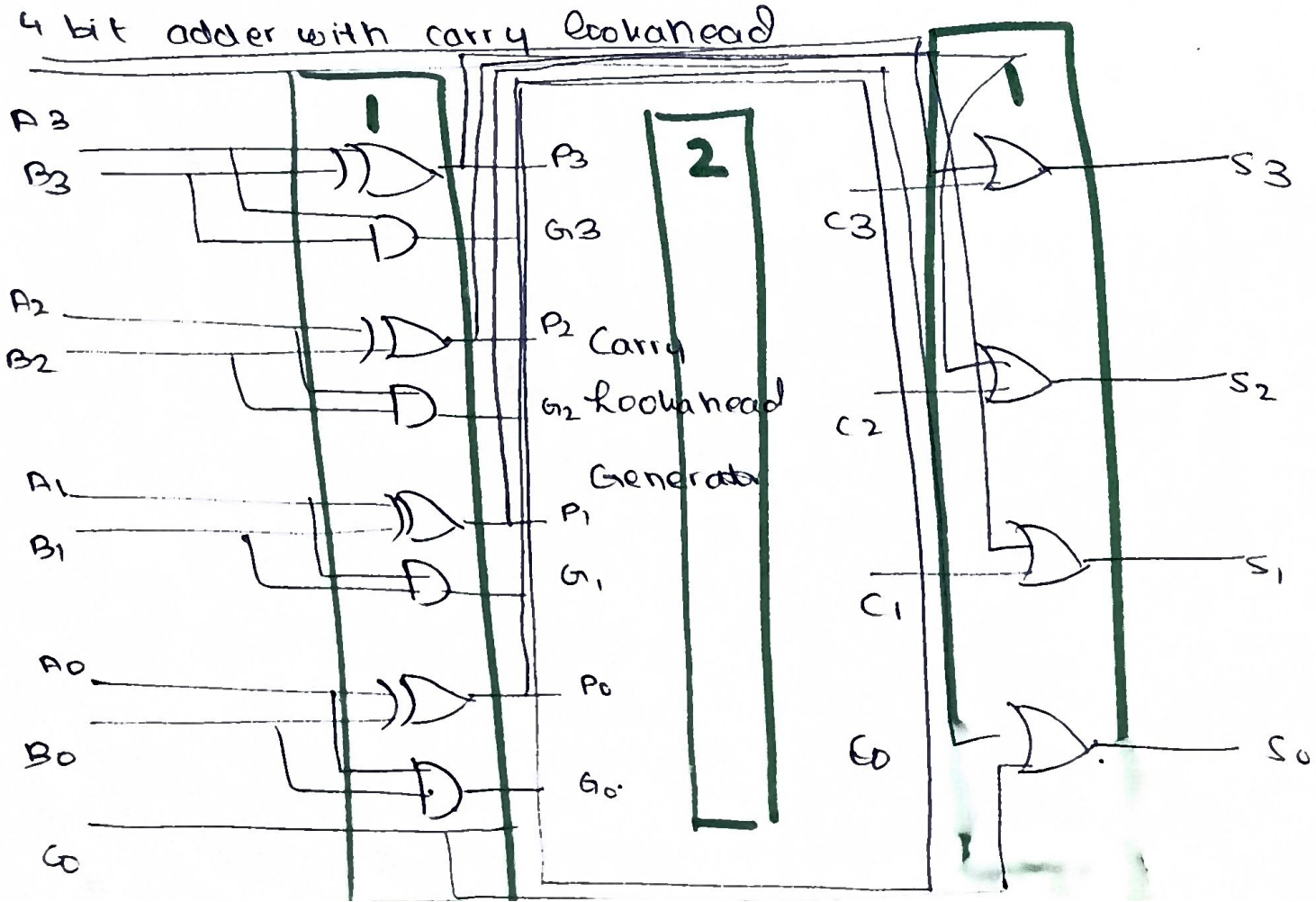
$C_3 = G_2 + P_2 C_2$

$= G_2 + P_2 (P_1 P_0 C_0 + P_1 G_0 + G_1)$

$= G_2 + P_2 P_1 P_0 C_0 + P_2 P_1 G_0 + P_2 G_1$.

Draw circuit diagram for C_1, C_2, C_3

* 4 bit adder with carry lookahead

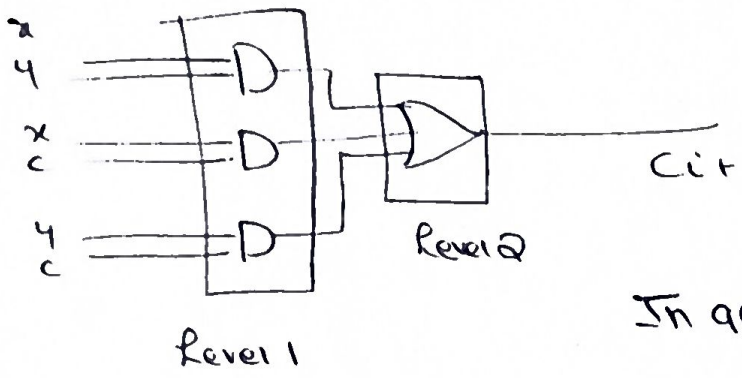


* Gate Delays

4-bit ripple carry adder

Sum = $x \oplus y \oplus z$

Carry = $xy + xc + yz$



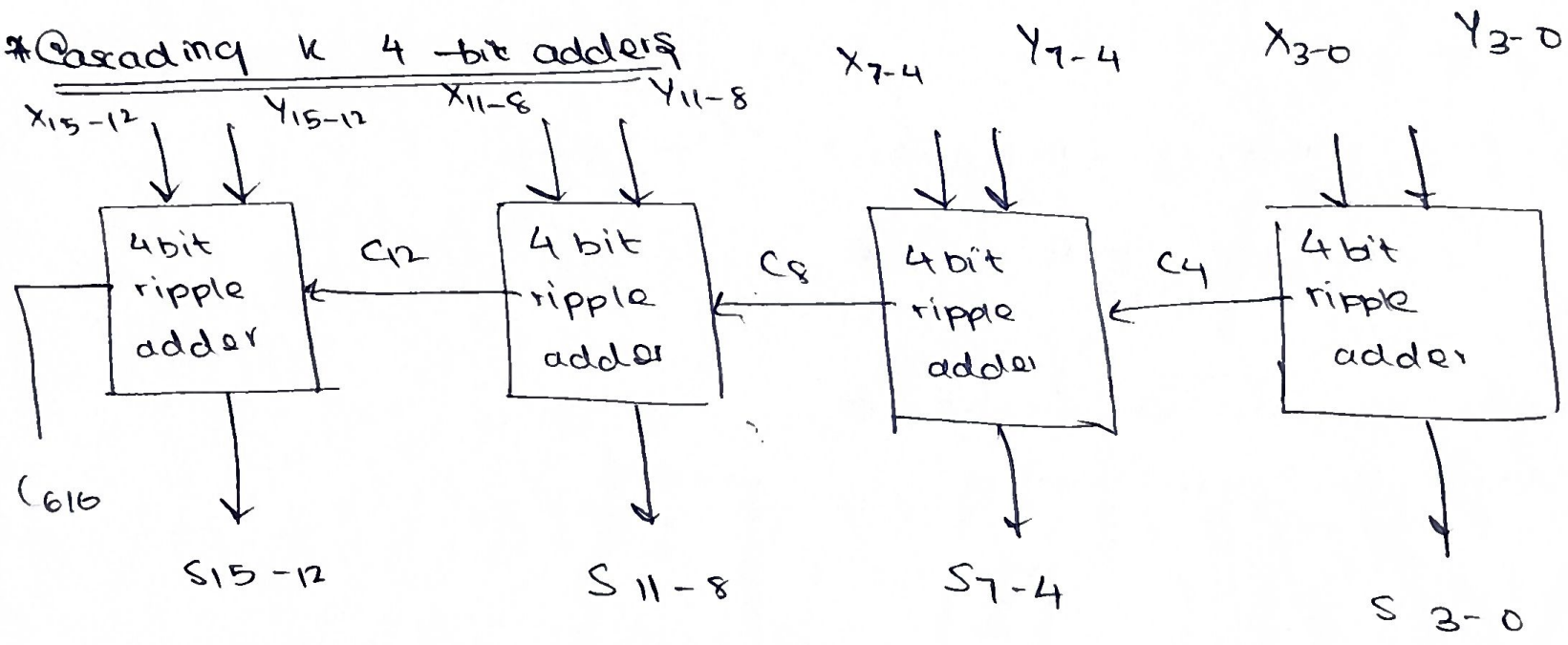
⇒ For each carry, there are 2 gate delays

In general, for an n bit ripple carry adder

$$c_n = n \times 2 \text{ delays}$$

$$s_n = 2n + 1 \text{ delays}$$

* Cascading k 4-bit adders



1 4 bit ripple adder has 8 gate delays

$C_4 = 8$

$C_8 = 16$

In general : $C_n = n \times 2^4$ gate delays

↳ ~~n = given no/4~~

$S_n = (n \times 4) + 1$

* CLA delays

→ All C_i's are generated in 2 gate delays

→ All P_is, G_is are generated 1 gate delay

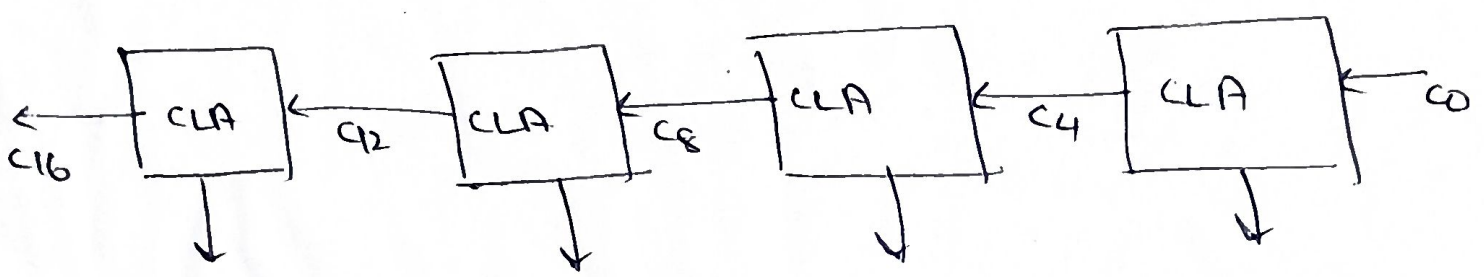
→ Sum generated in another gate delay

Total gate delay = 4

i.e Carry Output in 3 gate delays

Sum output in 4 gate delays

Can build a 16bit adder using a 4-bit CLA



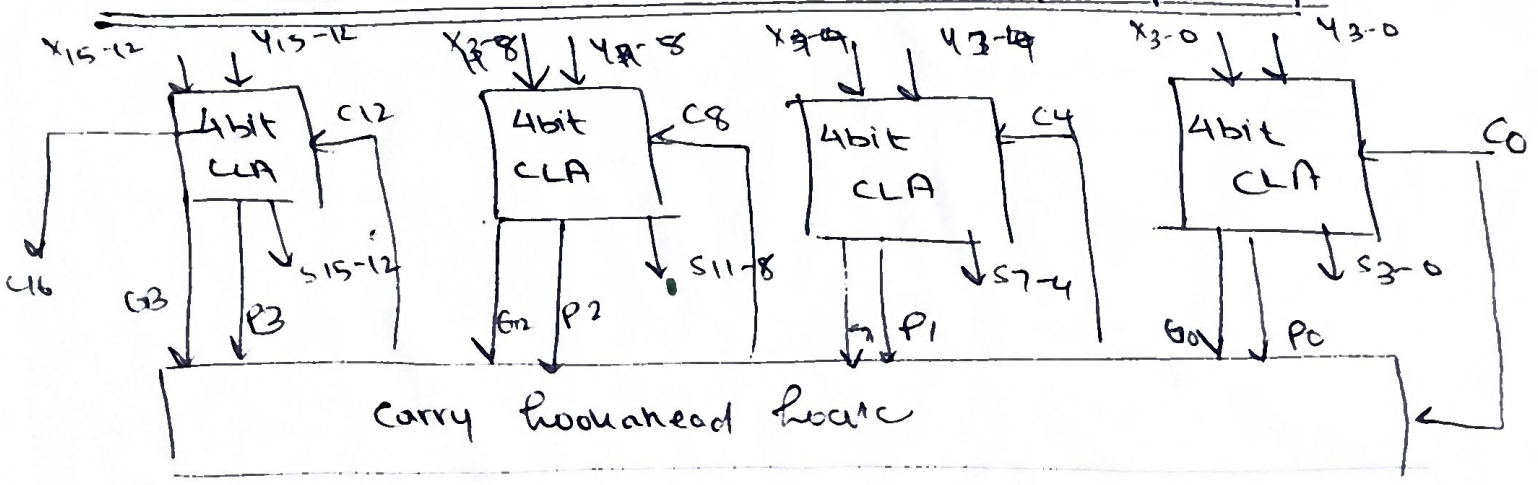
S_{12-15} $S_8 - S_{11}$ $S_4 - S_7$ $S_0 - S_3$

each carry op in 3 gate delays

⇒ Total gate delay = 12

each sum o/p in 4 gate delays ⇒ Total gate delay = 13

* 16 bit adder w/ 4 bit CLA as base w/o rippling



Here all carries are generated w/ the same delay

| Carry Gate Delay = 1 |

Check

$C_{16} = 5$

Unit - 2

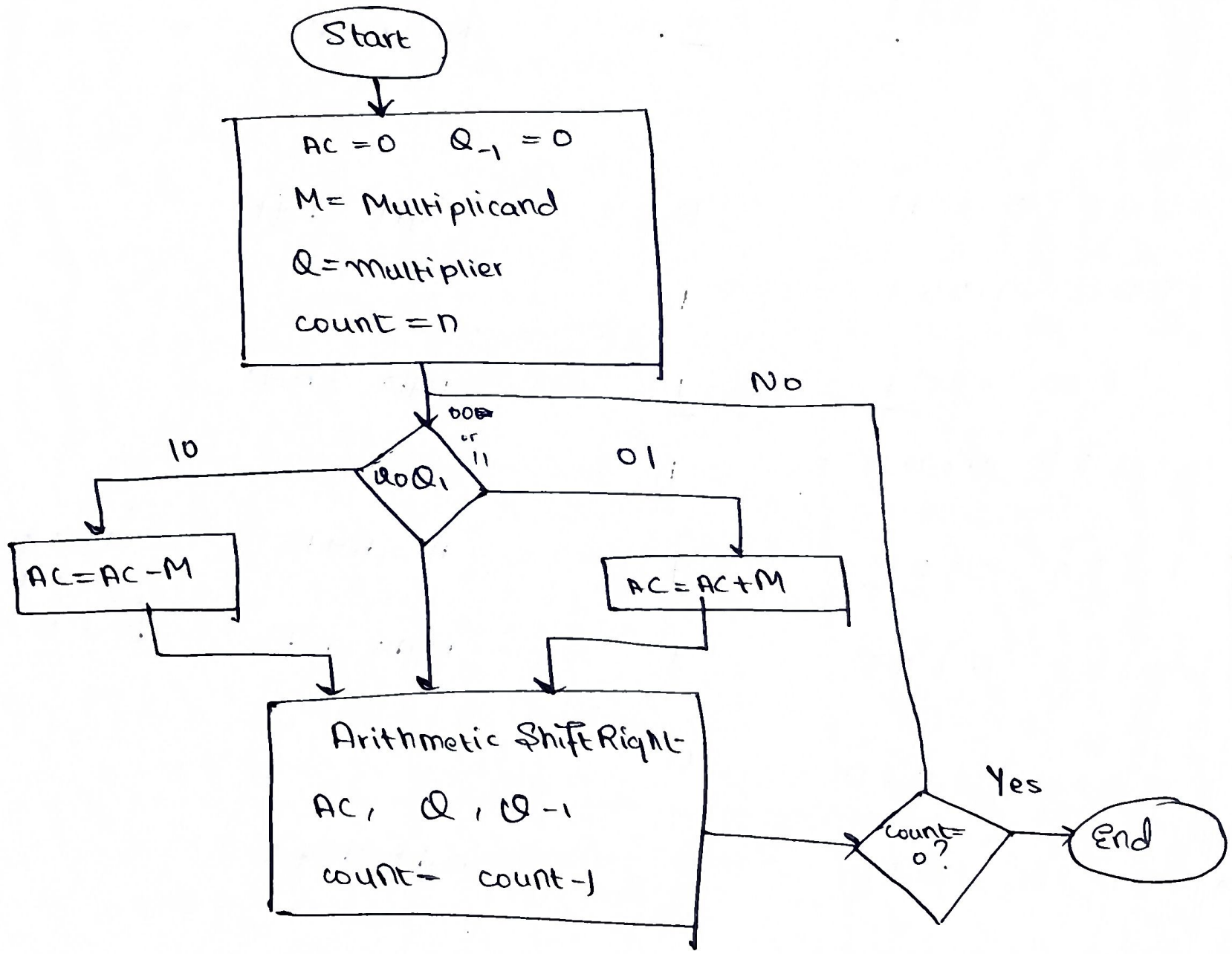
Computer Organization & Architecture

Multiplication & Division

A. MULTIPLICATION

* Booth Multiplication

Flowchart



Q1 7 x 3

M = 7 = 0111 (add 1 extra bit)

Q = 3 = 0011

-M = = 1001

0111

1st comp 1000

2's comp 1001

	A	Q_0	Q_1	operation
1)	0000	0011	0	A = A - M = 0000 + 1001 = 1001
2)	1001	0011	0	right shift
	1100	1001	1	
2)	1100	1001	1	right shift
	1110	0100	1	
3)	1110	0100	1	A = A + M = 1110 + 0111 = 10101
	0101	0100	1	right shift
	0010	1010	0	
4)	0010	1010	0	right shift
	0001	0101	0	

2^5 2^4 2^3 2^2 2^1 2^0

1 0 1 0 1

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

$= 2^4 + 2^2 + 2^0$

$= 16 + 4 + 1 = 21$

Q_2 : -13×-20

$M = -13 \Rightarrow M = 110011$

$-M = 001101$

$Q = -20 \Rightarrow Q = 101100$

$-Q = 010100$

	A	Q_0	Q_1	operation
①	000000	101100	0	right shift
	000000	010110	0	
②	000000	010110	0	right shift
	000000	001011	0	
③	000000	001011	0	A = A - M
				000000
				001101
				001101
	001101	001011	0	right shift
	~~100110~~	~~100101~~	~~1~~	
	000110	100101	1	
④	000110	100101	1	right shift
	000011	010010	1	
⑤	000011	010010	1	A = A + M
	110110	010010	1	000011
		~~0001~~	0	110011
	111011	001001	0	110110
				ASR
⑥	111011	001001	0	A = A - M
	001000	001001	0	111011
				001101
	000100	000100	1	1001000
				ASR

2 | 20
2 | 10 → 0
2 | 5 → 0
2 | 2 → 1
1 | 1 → .
0 → 1

1100

Q is -ve

⇓

consider extra bits when finding 2's complement

2 | 20
2 | 10 → 0
2 | 5 → 0
2 | 2 → 1
1 → 0

10100

0

-20 = 101011

+1 1

101100

1 1

101011

1

101100

2 | 13
2 | 6 → 1
2 | 3 → 0
2 | 1 → 1
0 → 1

01101

001101

110010

1

110011

000000

Ans : 1 0 0 0 0 0 0 0 0

2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

$= 2^8 + 2^2 + 2^1$

$= 256 + 4 + 2$

$= 260$

Ans = 260

2^8

Q3 -32×64

M = -32 => 1 00 000

=> 00 100000.

Q = 64 => 0 100 0000

	A	Q_0	Q_1	operation
①	0000 0000	0100 0000	0	shift right
	0000 0000	0010 0000	0	
②	0000 0000	0010 0000	0	shift right
	0000 0000	0001 0000	0	
③	0000 0000	0001 0000	0	shift right
	0000 0000	0000 1000	0	
④	0000 0000	0000 0100	0	shift right
	0000 0000	0000 0010	0	
⑤	0000 0000	0000 0010	0	shift right
	0000 0000	0000 0001	0	
⑥	0000 0000	0000 0001	0	A = A - M
	0010 0000	0000 0001	0	rightshift
	0001 0000	0000 0000	1	
⑦	~~0010 0000~~	~~0000 0001~~	~~0~~	A = A + M
	0001 0000	0000 0000	1	rightshift
	0011 0000	0000 0000	1	
	0001 1000	0000 0000	0	
⑧	0001 1000	0000 0000	0	right shift
	0000 1100 0000 0000		0	

-ve ans => take 2's complement

= 0100 0000 0000 = 2

64 is not -ve.

simply add extra 0s

2 | 32
2 | 16 → 0
2 | 8 → 0
2 | 4 → 0
2 | 2 → 0
2 | 1 → 0
0 → 1

-M

1 00 000

1 1 1 1 1 1

0 1 1 1 1 1

1

0 0 1 0 0 0 0 0

0000 0000
0010 0000
0010 0000

0001 0000
0010 0000
0011 0000 0

check

should be 2^{11}

$= -2048$

Q4. -22×-18

$M = -22 \Rightarrow M = 101010$

$-M = 010110$

$Q = -18$

$Q = 101110$

$\Rightarrow$

$-Q = 010010$

	A	Q	Q_0	Operation
①	000 000	101 110	0	AsR
	000 000	010 111	0	
②	000 000	010 111	0	$A = A - M$
	010 110	010 111	0	000 000 010 110 010 110
	001 011	001 011	1	AsR
③	001 011	001 011	1	AsR
	000 101	100 101	1	
④	000 101	100 101	1	AsR
	000 010	110 010	1	
⑤	000 010	110 010	1	$A = A + M$ 000 010 101 010 101 100
	101 100	110 010	1	AsR
	110 110	011 001	0	
⑥	~~110 000 011 001 0~~			$A = A\bar{+}M$ ~~111 110~~ ~~101 010~~ ~~100000~~
	~~110 000 001 100~~			AsR
	001 100	011 001	0	
	~~000 110 011 0~~			
	000 110	001 100	1	

000 110 001 100

$2^8\ 2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$2^8 + 2^7 + 2^3 + 2^2 = 256 + 128 + 8 + 4$

Ans = 396

$22 \to 11 \to 0,\ 5 \to 1,\ 2 \to 1,\ 1 \to 0,\ 0 \to 1$

10110

010110

10100

101001

101010

$18 \to 9 \to 0,\ 4 \to 1,\ 2 \to 0,\ 1 \to 0,\ 0 \to 1$

010010

10110

101110

Bit - Pair Multiplication

Multiplier bit - pair		multiplier bit on the right	multiplicand selected at i
i+1	i	i-1	
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

Ex1 -11 x 27

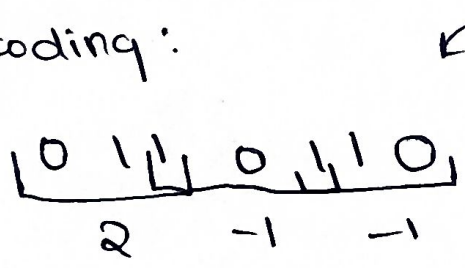
-11 → 1 10 1 0 1

27 → 0 11 0 1 1

i

extra bit

recoding:



1 1 0 1 0 1

2 -1 -1

0 0 1 0 1 1

1. write binary value of 27.
2. make binary of 11 as long as 27.
3. write binary value of -11
4. sign extend both by 1 bit

-ve ⇒ 1

+ve ⇒ 0

-1 ⇒ 2's complement

00 1 0 1 0

1

00 1 0 1 1

2 | 27

2 | 13 → 1

2 | 6 → 1

2 | 3 → 0

2 | 1 → 1

0 → 1

11011

11 = 1011

000 1011

111 0100

1

111 0 1 0 1

1011

0100

1

0101

01011

10100

1

10101

P.T.O

1 1 0 1 0 1

2 −1 −1

sign extend

0 0 0 0 0 0 | 0 0 1 0 1 1

0 0 0 0 0 0 1 0 1 1

1 1 1 0 1 0 1 0

1 1 1 0 1 1 0 1 1 1

find 2's complement

2^{10} 2^9 2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

0 0 0 1 0 0 1 0 1 0 0

0 0 0 1 0 0 1 0 0 1 0

$2^8 + 2^5 + 2^4$ $2^8 + 2^5 + 2^3 + 2^0$

$= 256 + 32 + 16$ $256 + 32 + 8 + 1$

$= -297$

(ii) −1

⇒ 2's complement

(iii) × 2

⇒ × 10

1 1 0 1 0 1

1 0

0 0 0 0 0 0

1 0 1 0 1 0

1 1 0 1 0 1 0

Q2 13 × −6

13 → 1 1 0 1

6 → 0 1 1 0

−6 → 1 0 1 0

add sign bit 13 → 0 1 1 0 1

1 1 0 1 0

2 | 6

2 | 3 → 0

2 | 1 → 1

0 → 1 0110

1 0 0 1

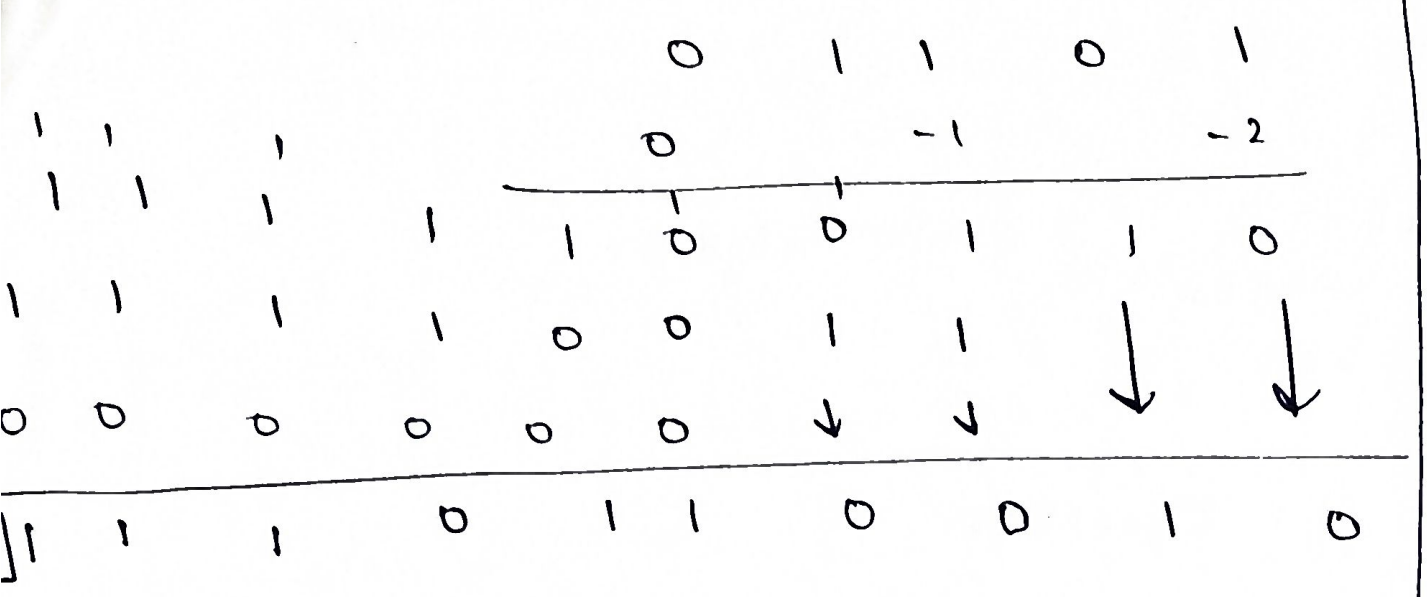
1 0 1 0

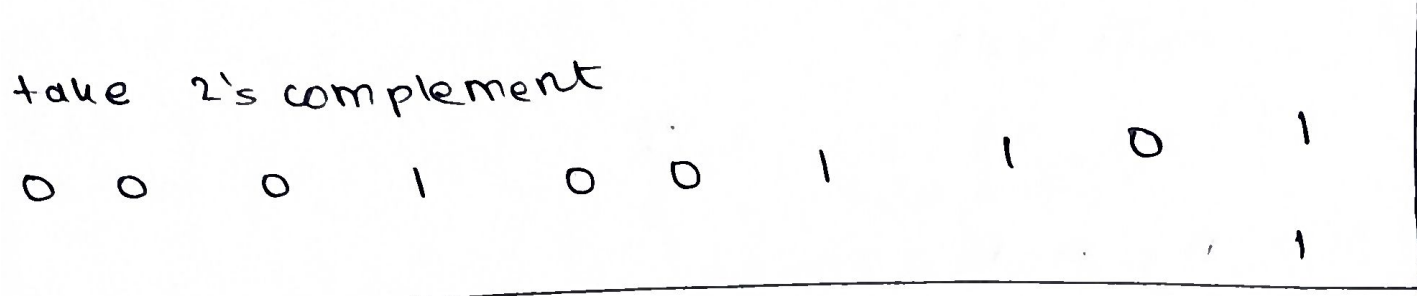
recoding

1 1 1 0 1 0 0 add 0

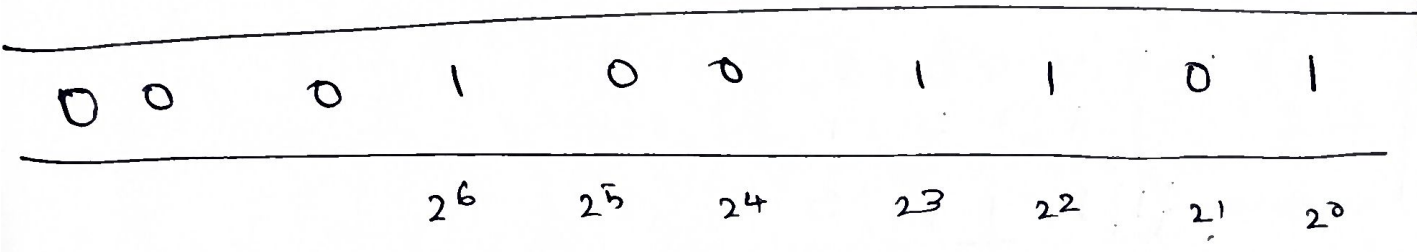
0 −1 −2

extend sign to make 3





take 2's complement



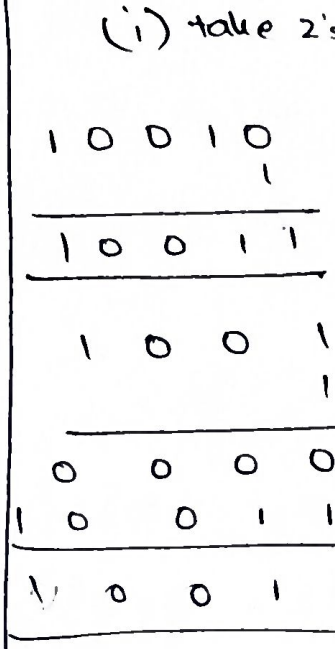
$2^6 + 2^3 + 2^2 + 2^0$

$64 + 8 + 4 + 1$

$72 + 6 = -78$

Ans = -78

(i) take 2's complement



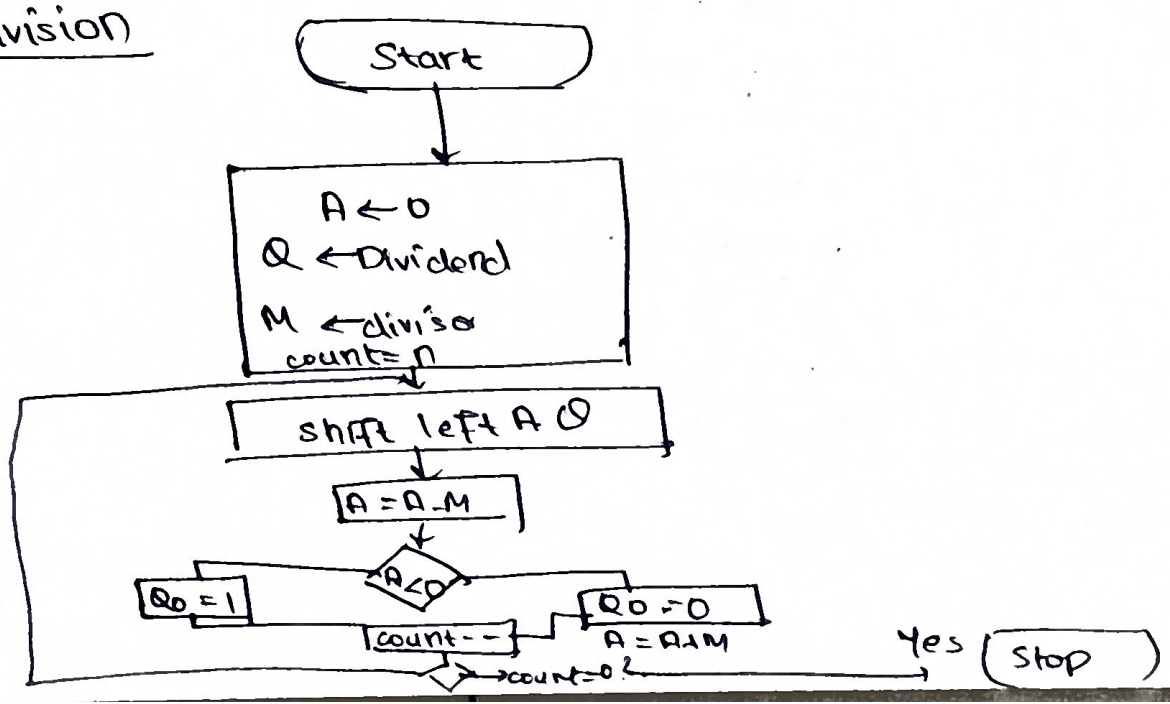
(ii) -1 ⇒ 2's complement

= 10011

13 / 6 / 7 / 6

Division

A. Restoring Division



Example1 $10 \div 3$ 1100

$Q = 1010$

$B = 0011$ $\quad -B = 1101$

add 1 bit

$A = 00000$

A	Q	Operation	n
00000	1010	shift left	4
00001	010 0		

$A \leftarrow A - B$

```
  00001
  11101
 ------
  11110
```

restore

$A \leftarrow A + B$

```
  11110
  00011
 ------
 1|00001
```

00001	0100	shift left	3
00010	1000		

$A \leftarrow A - B$

```
  00010
  11101
 ------
  11111
```

restore

$A \leftarrow A + B$

```
  11111
  00011
 ------
 1|00010
```

A	Q	Operation	n
00010	1000	shift left	
00101	000 1		2.

A ← A - B

```
 00101
 11101
10 0010
```

00010	000 1	shift left	
~~01010~~	001 1		
00100			

A ← A - B

```
 01010
 11101
10 00111
```

A	Q
00001	0011
Remainder	Quotient

$10 \div 3 : Q = 3, \; R = 1$

Eg Q 14 % 3

$Q = 14 = 1110$

$B = 3 = 0011$

$B = 00011$

$-B = 11101$

A	Q	Operation	n
00000	1110	Shift left	4
00001	110 0	A ← A - B	
		00001	
		11101	
		1 1110	
		restore	
		A ← A + B	
		00001	
		11110	
		1 00001	
00001	1100		

A	Q	Operation	n
0 0 0 0 1	1 1 0 0	shift left	3
0 0 0 1 1	1 0 0 0	A ← A − B 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0	
0 0 0 0 0	1 0 0 0	shift left	2
0 0 0 0 1	1 0 0 0	A ← A − B 0 0 0 0 1 1 1 1 0 1 1 1 1 0 1 restore A ← A + B	
0 0 0 0 1	0 0 0 0	shift left	1
0 0 0 1 0	0 0 0 1	A ← A − B 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1	

- Quotient :- 0011 ⇒ 3

  Remainder : 0010 ⇒ 2

14 ÷ 3 ≈ 3, remainder = 2

Non-Restoring Division

Steps

A ← 0

B ← Divisor

Q ← Dividend

1. Shift left A & Q
2. If A = 0 ⇒ do A ← A − B
   else do A ← A + B
3. Repeat n times

Ex1 10 ÷ 3

A = 0

Q = 1010

B = 0011

$-B = 11101$

A	Q	Operation	n
00000	1010	shift left	4
00001	0100	A ← A − B: 00001 + 11101 = 11110	
11110	1000	shift left; A ← A + B: 11110 + 00011 = 00001	3
00001	0001	shift left; A ← A + B: 00001 + 11101 = 11110	2
11110	0010		

A	Q	Operation	n
00010	0001	shift left	1
00100	0011	A ← A - B 00100 11101 00001	
00001	0011		

Quotient = 0011 = 3

Remainder = 00001 = 1

Booth's Multiplication

1. Find larger no.
2. add a 0 to MSB
3. For the smaller no, make it the same size as the larger no
4. If there nos are -ve, find the 2's complement
5. Recode the 2nd no., (add a 0 at the end)

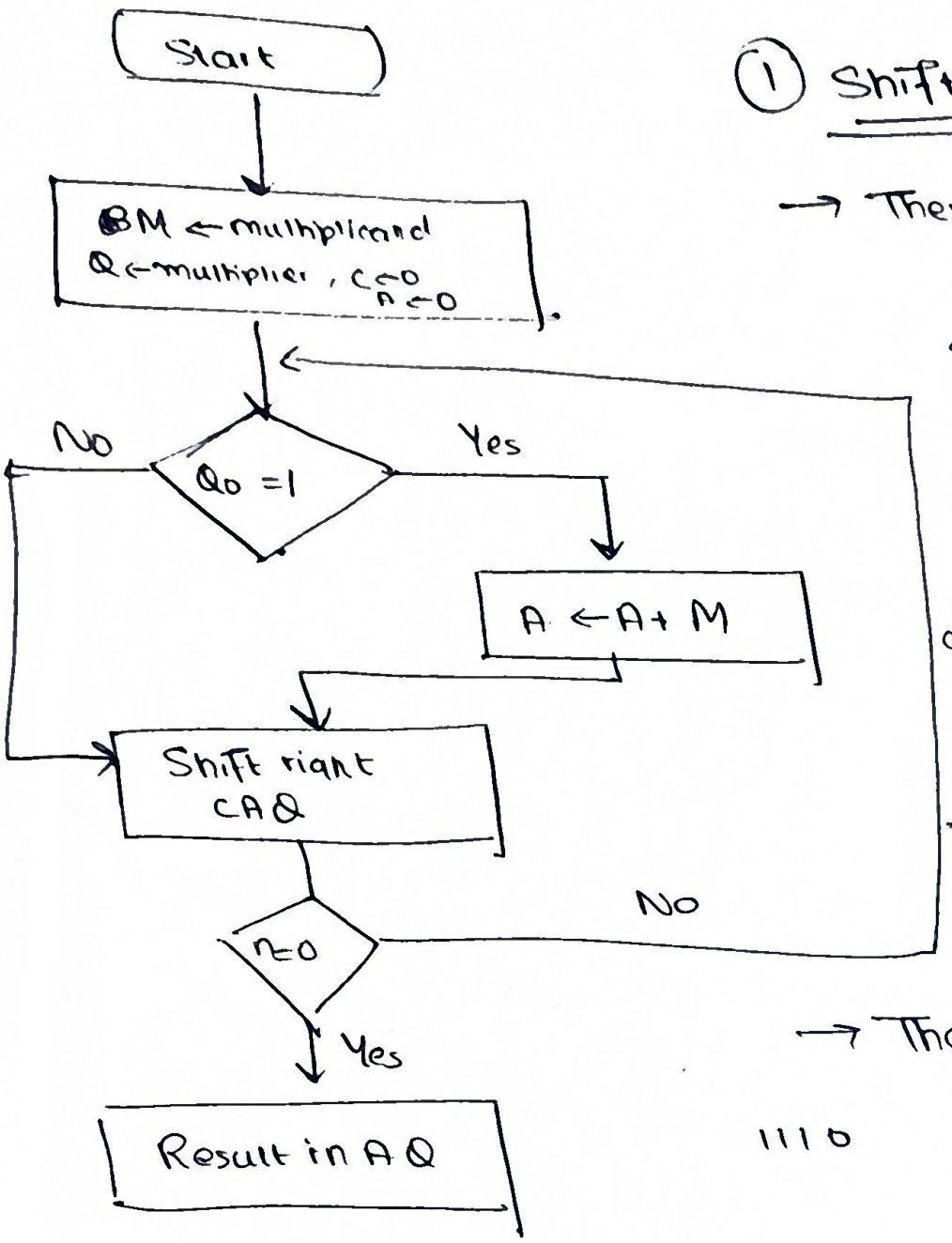
* Multiplication

Shift and add

array multiplier

booth's multiplication

Algorithm



① Shift and Add

→ There is the multiplicand & multiplier, the carry & accumulator are initialized to 0

→ If the LSB of the multiplier is 1, add the multiplicand to the accumulator

→ Otherwise shift right CAQ.

→ If the LSB is 0, again shift otherwise add

→ The output is in AQ.

Example 14 × 10

M ← 14 = 1110

Q ← 10 = 1010

C	A	Q	Operation
0	0000	1010	~~1~~ shift
0	0000	0101	② add
0	1110	0101	shift
0	0111	0010	③ shift
0	0011	1001	④ add
1	0001	1001	shift
0	1000	1100	

128 + 8 + 4 = 140

1 1
0011
1110
10001

Example 11 × 13 -

M = 1011

M ← 11

Q ← 13

C	A	Q	Operation
0	0000	1101	Initialization
0	1011	1101	① add.
0	0101	1110	shift
0	0010	1111	② shift
0	1101	1111	③ add
0	0110	1111	shift
1	0001	1111	④ add
0	1000	1111	shift right

1011
0000
0010
1011
1101
1101
0110
1011
10001

(2) Array Multiplier

b_1 b_0

a_1 a_0

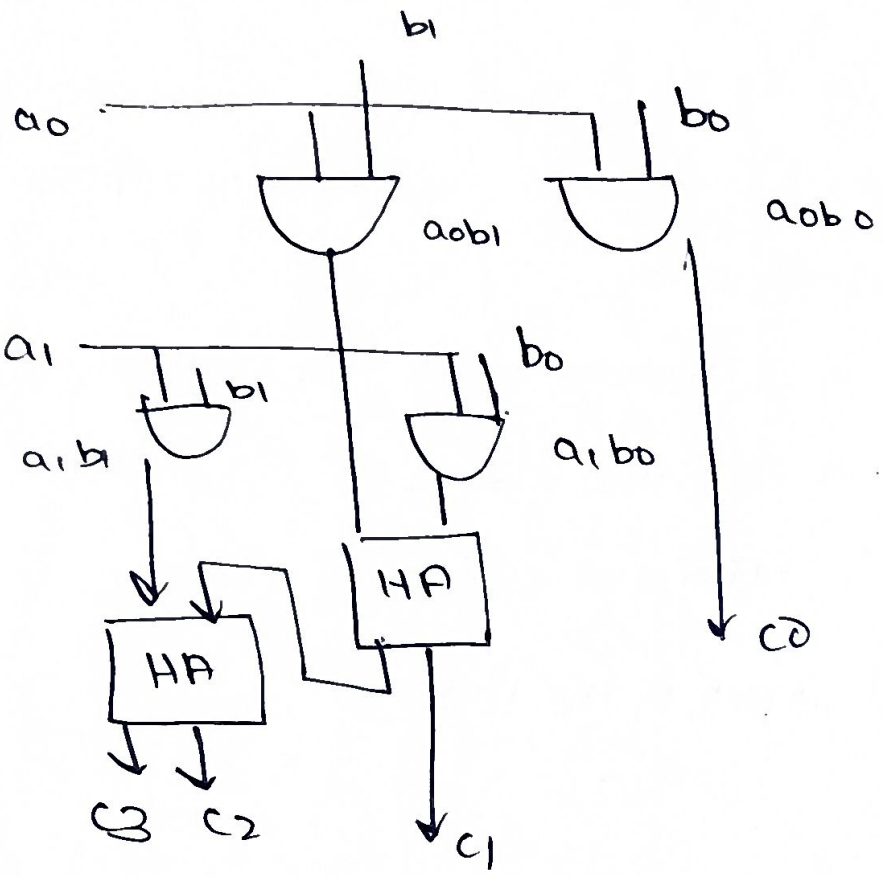
---

a_0b_1 a_0b_0

a_1b_1 a_1b_0

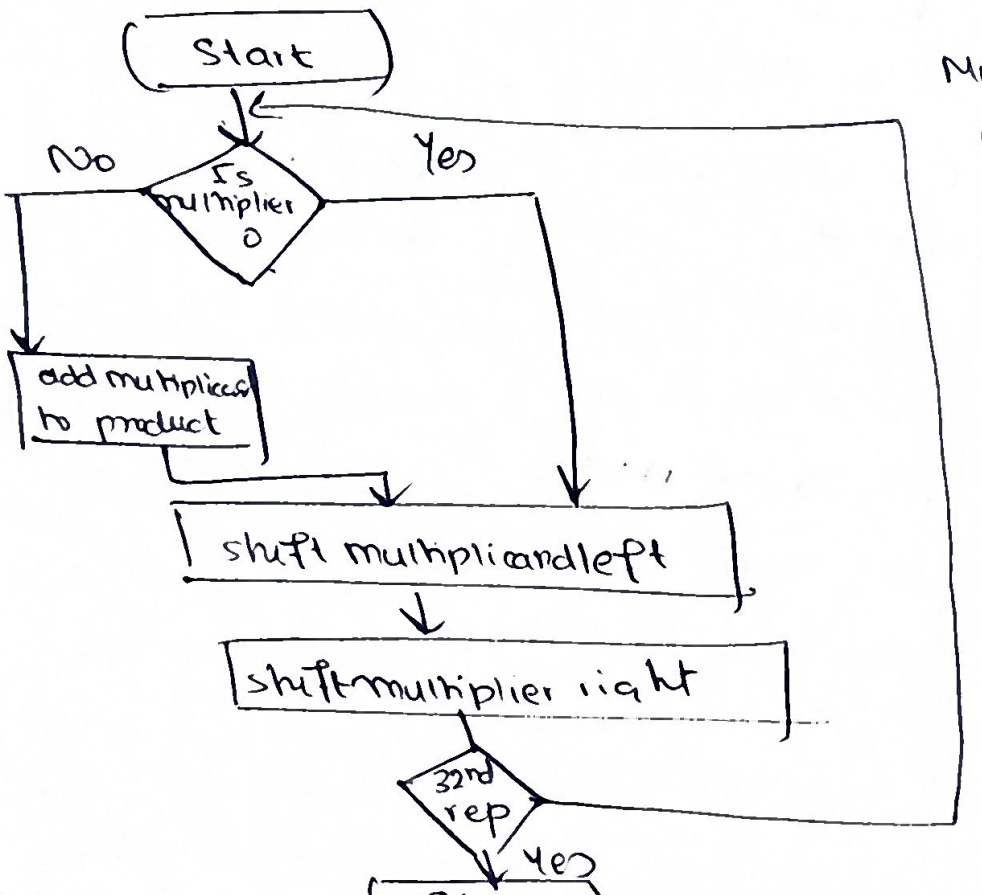
---

c_3 c_2 c_1 c_0



Check implementation
add multiplicand to the left half of the product
shift multiplier right

(3) Left shift (Multiplicand shift left / Multiplier shift right)

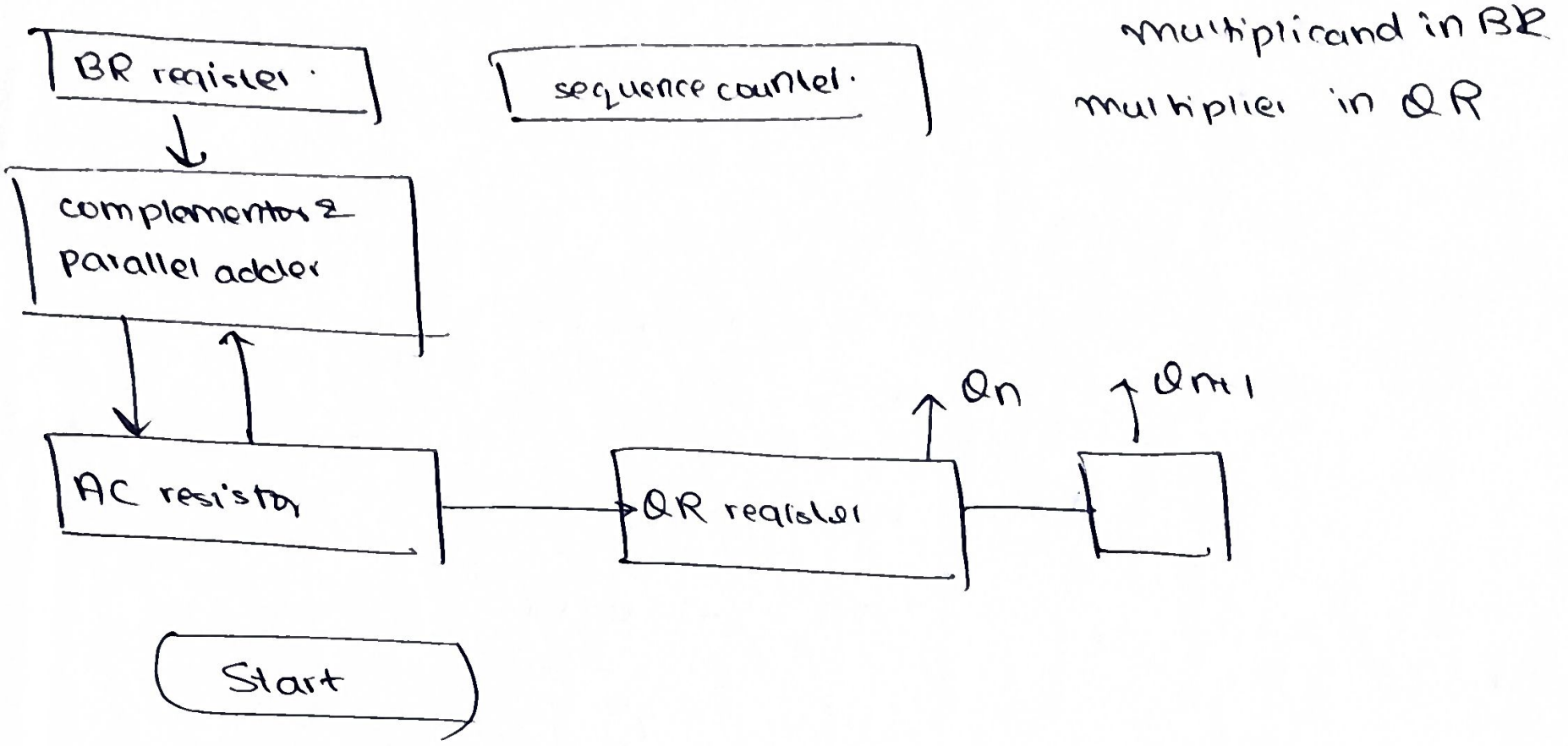


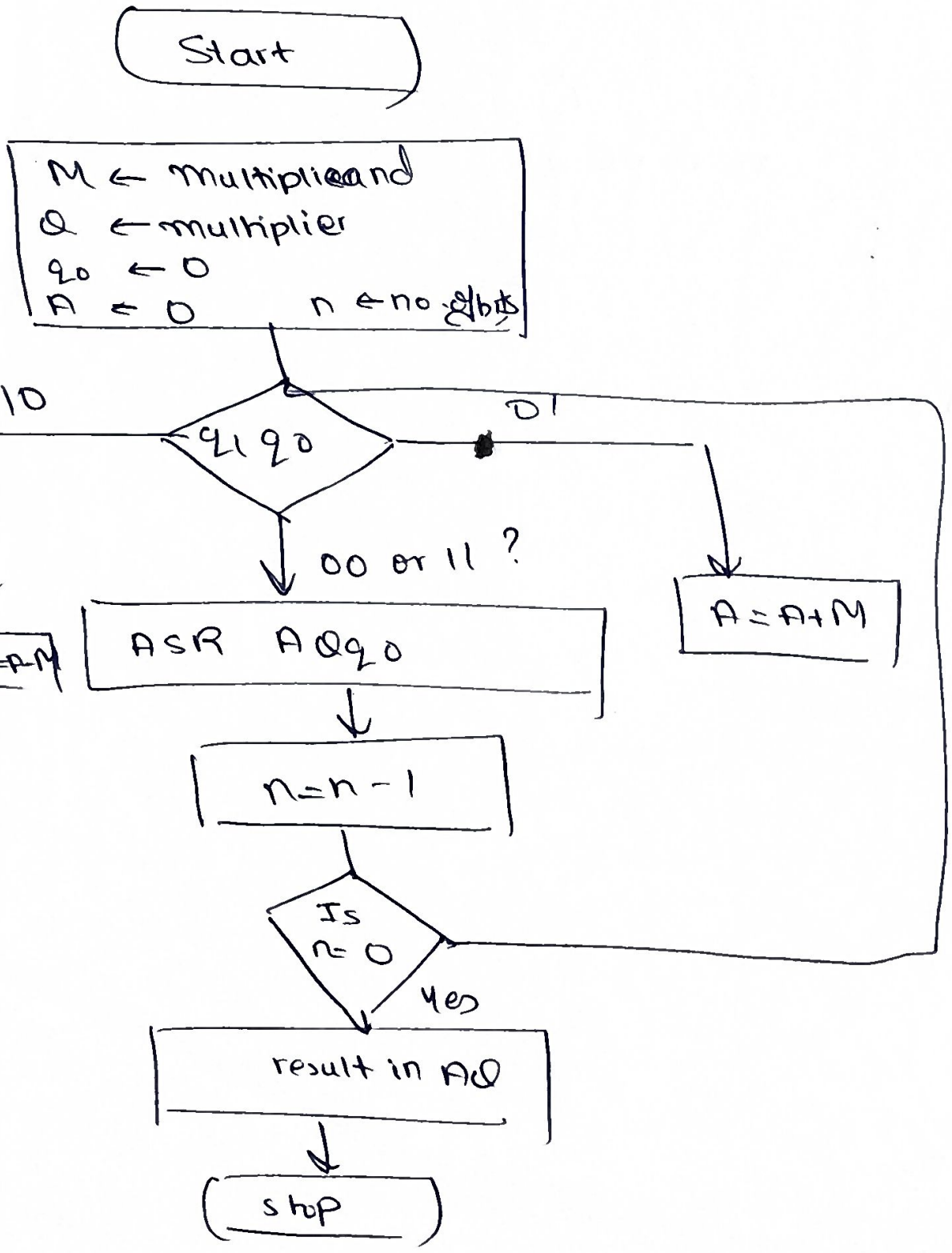
Ex 0010 x 0110

Multiplier	Multiplicand	Product	Op	
0110	0010	0000	init	
0011	0100	0000	slr	1
0001	0100	0100	add	2
0000	1000	0100	slr	
0001	1000	1100	add	3
0000	0000	1100	slr	4
0000	0000	1100	slr	

(4) Booth's Algorithm

Hardware





Example

$-7 \times +3$

↓ Multiplier

↓ multiplicand

-7

$= 2$'s complement of 0111

$= 1001_2$

~~$3 = 2$'s complement of 0011~~

n	A	Q	q_0	Operation
4	0000	0011	0	Initialization
	0111	0011	0	$10 \Rightarrow A = A - M$
	0011	1001	1	ASR AQq0
3	0001	1100	1	ASR AQq0
2	1010	1100	1	$01 \Rightarrow A = A + M$
	~~0101~~	~~0110~~	~~0~~	
	~~1011~~	~~0110~~	~~0~~	
1	~~1011~~ 1101	1011	0	ASR AQq0
	~~1101~~	~~1011~~	~~0~~	ASR AQq0
sf	1101	1011	0	
0	1110	1101	1	

find 2's complement

$M = 1001$

$-M = 0111$

$$\begin{array}{r} 0001 \\ 1001 \\ \hline 1010 \end{array}$$