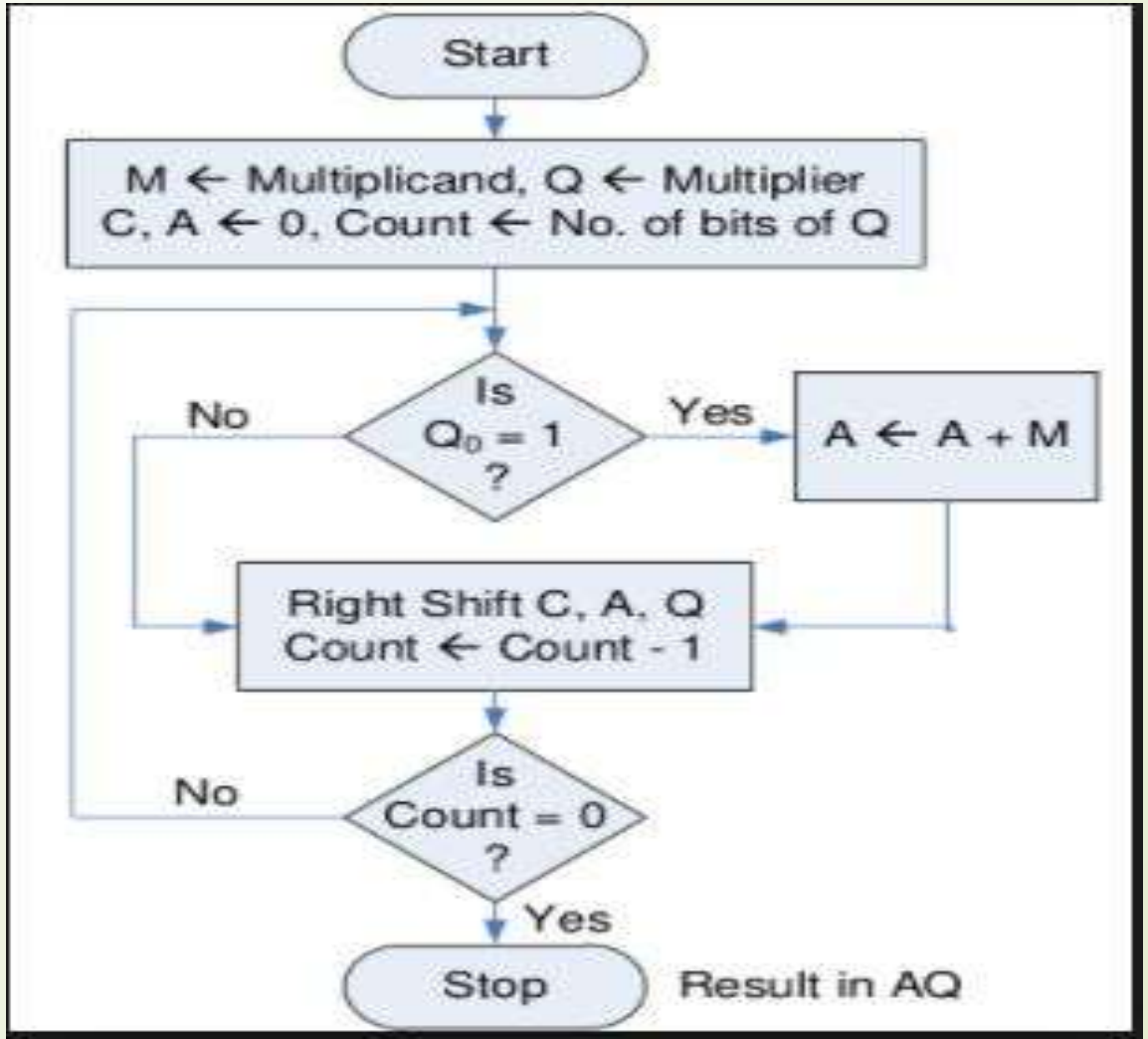
CHAPTER-6

Arithmetic Unit

Shift-add multiplication:

=> multiplication of unsigned numbers.

=> multiplication can be performed by repeated addition of multiplicand and shifting



Contd...

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

RTL code for shift-add multiplication

```
    1:    C ← 0, U ← 0, i ← n
Y0  2:    CU ← U + X
    2:    i ← i − 1
    3:    shr(CUV), cir(Y)
Z'  3:    GOTO 2
Z   3:    FINISH ← 1
```

Contd...

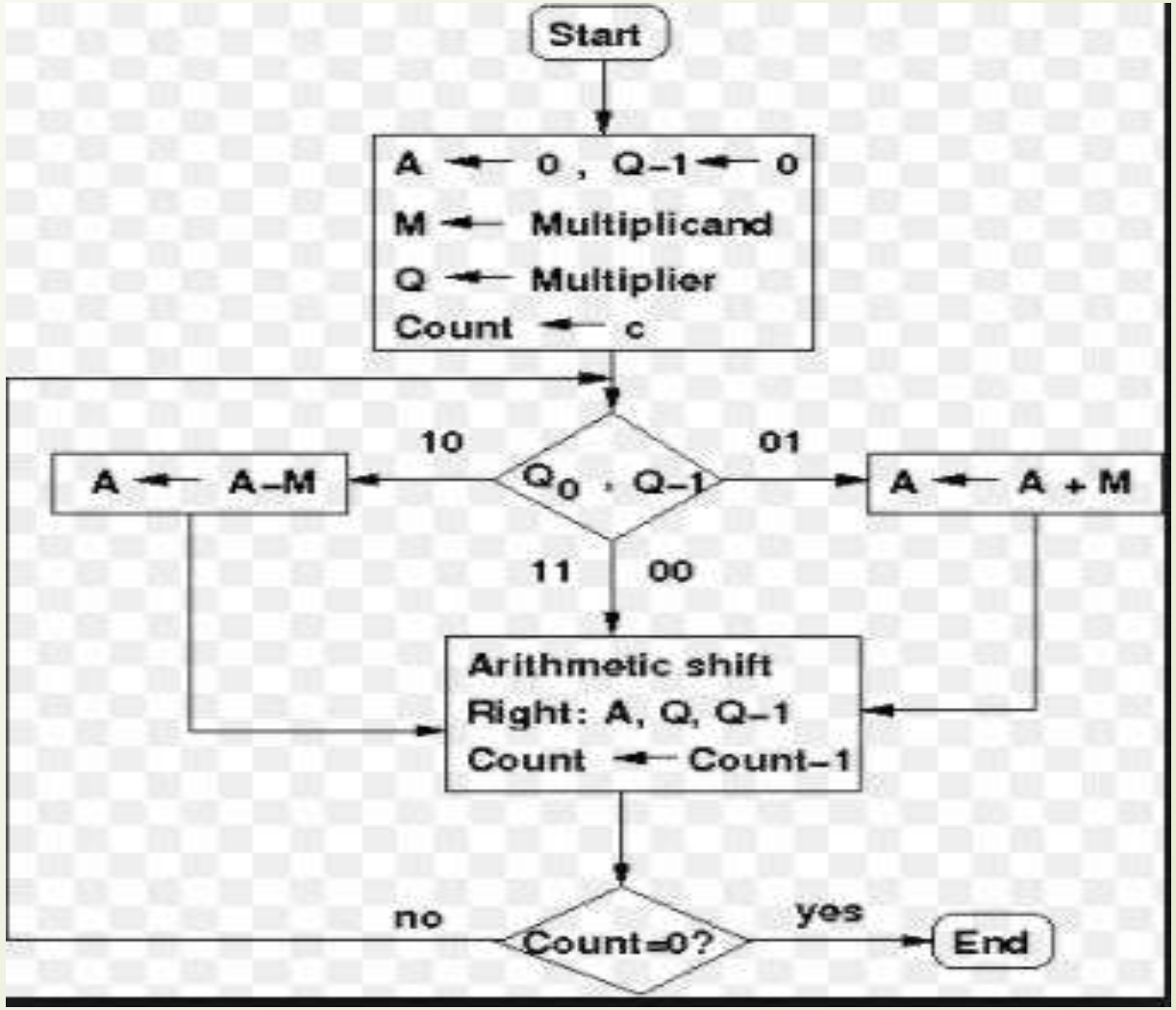
Example UV <- X x Y; X= 1101 and Y=1011.

Conditions	Micro-operations	i	C	U	V	Y	Z	FINISH
START		x	x	xxxx	xxxx	1011		0
1	$C \leftarrow 0, U \leftarrow 0, i \leftarrow 4$	4	0	0000			0	
$Y_0$2,2	$CU \leftarrow U + X$, $i \leftarrow i - 1$	3	0	1101			0	
3,Z'3	shr(CUV), cir(Y), GOTO 2		0	0110	1xxx	1101		
$Y_0$2,2	$CU \leftarrow U + X$, $i \leftarrow i - 1$	2	1	0011			0	
3,Z'3	shr(CUV), cir(Y), GOTO 2		0	1001	11xx	1110		
2	$i \leftarrow i - 1$	1					0	
3,Z'3	shr(CUV), cir(Y), GOTO 2		0	0100	111x	0111		
$Y_0$2,2	$CU \leftarrow U + X$, $i \leftarrow i - 1$	0	1	0001			1	
3,Z3	shr(CUV), cir(Y), FINISH←1		0	1000	1111	1011		1

Booth Algorithm

- It reduces the number of addition.
- Negative multiplier, positive multiplier are treated the same.
- For negative numbers multiplication, the operation is performed by representing the negative numbers in it's 2's complement form.
- ADVANTAGES:
  =>When there are lot of ones, all will be cancelled and replaced with zeros.
  => More zeros means, it reduces the addition(no operation is done with zeros)
  => But shift remains the same, can't be reduced.

Contd...


Start
A ← 0 , Q–1 ← 0
M ← Multiplicand
Q ← Multiplier
Count ← c
10
01
A ← A–M
Q_0 , Q–1
A ← A + M
11
00
Arithmetic shift
Right: A, Q, Q–1
Count ← Count–1
no
Count=0?
yes
End

Multiply 9 x -3 using booth algorithm.

$(9)_{10} = (01001)_2$

$\therefore M = 01001$

$(3)_{10} = (00011)_2$

$\therefore (-3)_{10} = 11101 = Q$

A	Q	Q_{-1}	Add(M)	Sub($\bar{M}+1$)	Count	Remarks
00000	11101	0	01001	10111	5	Initialization
10111	11101	0	–	–	4	A←A-M
11011	11110	1	–	–	4	ASr AQQ$_{-1}$ Count←count-1
00100	11110	1	–	–	–	A←A+M
00010	01111	0	–	–	3	ASr AQQ$_{-1}$ Count←count-1
11001	01111	0	–	–	–	A←A-M
11100	10111	1	–	–	2	ASr AQQ$_{-1}$ Count←count-1
11110	01011	1	–	–	1	ASr AQQ$_{-1}$ Count←count-1
11111	00101	1	–	–	0	ASr AQQ$_{-1}$ Count←count-1

Contd...

- Result Verification:1 1 1 1 1 0 0 1 0 1
  Since there is no end around carry, we find it's 2's complement and place '-' sign before the equivalent decimal.
  i.e. 0 0 0 0 0 1 1 0 1 1
  i.e. -27

RTL code for Booth algorithm

1: U <- 0, Y_{-1} <- 0, I <- n

Y_0Y_{-1}'2: U <- U + X' + 1

Y_0'Y_{-1}2: U <- U + X

2: i <- i – 1

3: ashr(UV), cir(Y), Y_{-1} <- Y_0

Z'3: goto 2

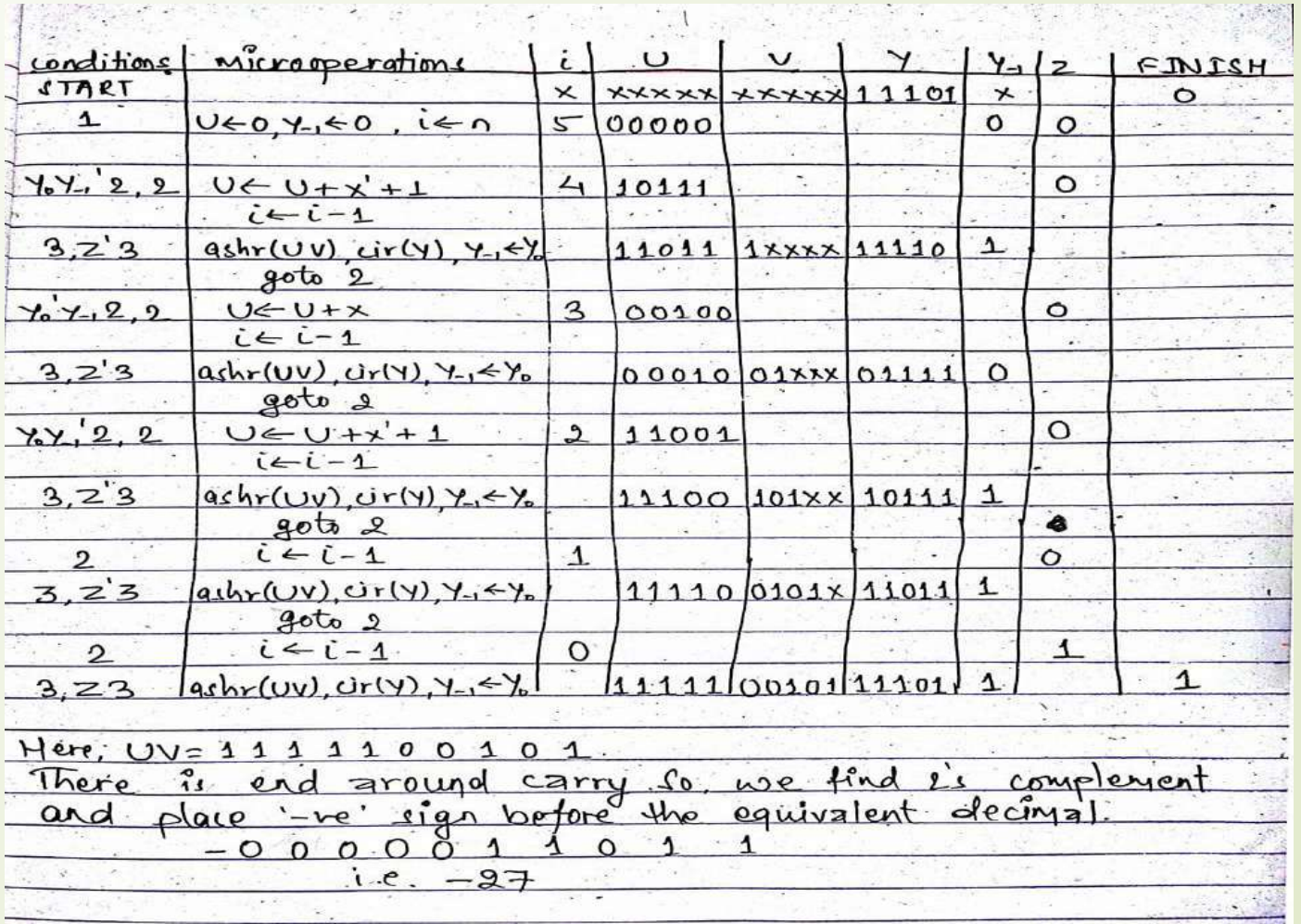
Z3: FINISH <- 1

Perform 9 x -3 multiplication using Booth algorithm.

Here,

X = 9 = 01001

Y = -3= 11101



conditions	microoperations	i	U	V	Y	Y_{-1}	Z	FINISH
START		x	xxxxx	xxxxx	11101	x		0
1	$U \leftarrow 0, Y_{-1} \leftarrow 0$, $i \leftarrow n$	5	00000			0	0	
$Y_0 Y_{-1}'$ 2, 2	$U \leftarrow U + x' + 1$ $i \leftarrow i - 1$	4	10111				0	
3, Z'3	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$ goto 2		11011	1xxxx	11110	1		
$Y_0' Y_{-1}$ 2, 2	$U \leftarrow U + x$ $i \leftarrow i - 1$	3	00100				0	
3, Z'3	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$ goto 2		00010	01xxx	01111	0		
$Y_0 Y_{-1}'$ 2, 2	$U \leftarrow U + x' + 1$ $i \leftarrow i - 1$	2	11001				0	
3, Z'3	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$ goto 2		11100	101xx	10111	1		
2	$i \leftarrow i - 1$	1					0	
3, Z'3	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$ goto 2		11110	0101x	11011	1		
2	$i \leftarrow i - 1$	0					1	
3, Z3	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$		11111	00101	11101	1		1

Here, UV = 1 1 1 1 1 0 0 1 0 1.
There is end around carry so, we find 2's complement and place '-ve' sign before the equivalent decimal.

-0 0 0 0 0 1 1 0 1 1

i.e. -27

Shift subtract division

(Before we look at the operation of the entire algorithm, consider the circumstances under which the first step terminates the algorithm. For 112÷7 with n=4, UV= 0111 0000 and X= 0111. Since, U≥X, the algorithm exists immediately. Had it proceeded, it should have produced a result of 16 (1 0000) and a remainder of 0. The value 1 0000 cannot be stored in a 4-bit register), this is the reason for the overflow.

$$
\begin{aligned}
G\mathbf{1}: &\quad FINISH \leftarrow 1,\ OVERFLOW \leftarrow 1 \\
\mathbf{2}: &\quad Y \leftarrow 0,\ C \leftarrow 0,\ OVERFLOW \leftarrow 0,\ i \leftarrow n \\
\mathbf{3}: &\quad \mathrm{shl}(CUV),\ \mathrm{shl}(Y),\ i \leftarrow i - 1 \\
(C + G)\mathbf{4}: &\quad Y_0 \leftarrow 1,\ U \leftarrow U + X' + 1 \\
Z'\mathbf{4}: &\quad \text{GOTO } \mathbf{3} \\
Z\,\mathbf{4}: &\quad FINISH \leftarrow 1
\end{aligned}
$$

Contd...

Trace of the RTL code for the operation 147 ÷ 13.
Here, U = 1001, V = 0011, X = 1101 and n=4.

Trace of the RTL code for the shift-subtract division algorithm

Conditions	Micro-operations	i	C	U	V	Y	Z	FINISH
START		x	x	1001	0011	xxxx		0
1	NONE							
2	$Y \leftarrow 0$; $C \leftarrow 0$, $OVERFLOW \leftarrow 0$, $i \leftarrow 4$	4	0			0000	0	
3	shl(CUV), shl(Y), $i \leftarrow i - 1$	3	1	0010	0110	0000	0	
$(C + G)4, Z'4$	$Y_0 \leftarrow 1$, $U \leftarrow U + X' + 1$, GOTO 3			0101		0001		
3	shl(CUV), shl(Y), $i \leftarrow i - 1$	2	0	1010	1100	0010	0	
$Z'4$	GOTO 3							
3	shl(CUV), shl(Y), $i \leftarrow i - 1$	1	1	0101	1000	0100	0	
$(C + G)4, Z'4$	$Y_0 \leftarrow 1$, $U \leftarrow U + X' + 1$, GOTO 3			1000		0101		
3	shl(CUV), shl(Y), $i \leftarrow i - 1$	0	1	0001	0000	1010	1	
$(C + G)4, Z4$	$Y_0 \leftarrow 1$, $U \leftarrow U + X' + 1$, $FINISH \leftarrow 1$		~~0100~~	0100	~~1011~~	1011		1

Binary Coded Decimal(BCD)

Binary Coded Decimal BCD Numbers

Each decimal digit to be represented in BCD is converted into its 4 digit binary equivalent. The final number will consist of four times the number of original decimal digits.

As the decimal digits relating to 1010, 1011, 1100, 1101, 1110, 1111 do not exist these are invalid.

Decimal	Binary	BCD
0	0000 0000	0000 0000
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0011	0000 0011
4	0000 0100	0000 0100
5	0000 0101	0000 0101
6	0000 0110	0000 0110
7	0000 0111	0000 0111
8	0000 1000	0000 1000
9	0000 1001	0000 1001
10	0000 1010	0001 0000
11	0000 1011	0001 0001
12	0000 1100	0001 0010
13	0000 1101	0001 0011
14	0000 1110	0001 0100
15	0000 1111	0001 0101
16	0001 0000	0001 0110
17	0001 0001	0001 0111
18	0001 0010	0001 1000
19	0001 0011	0001 1001
20	0001 0100	0010 0000

BCD adder:

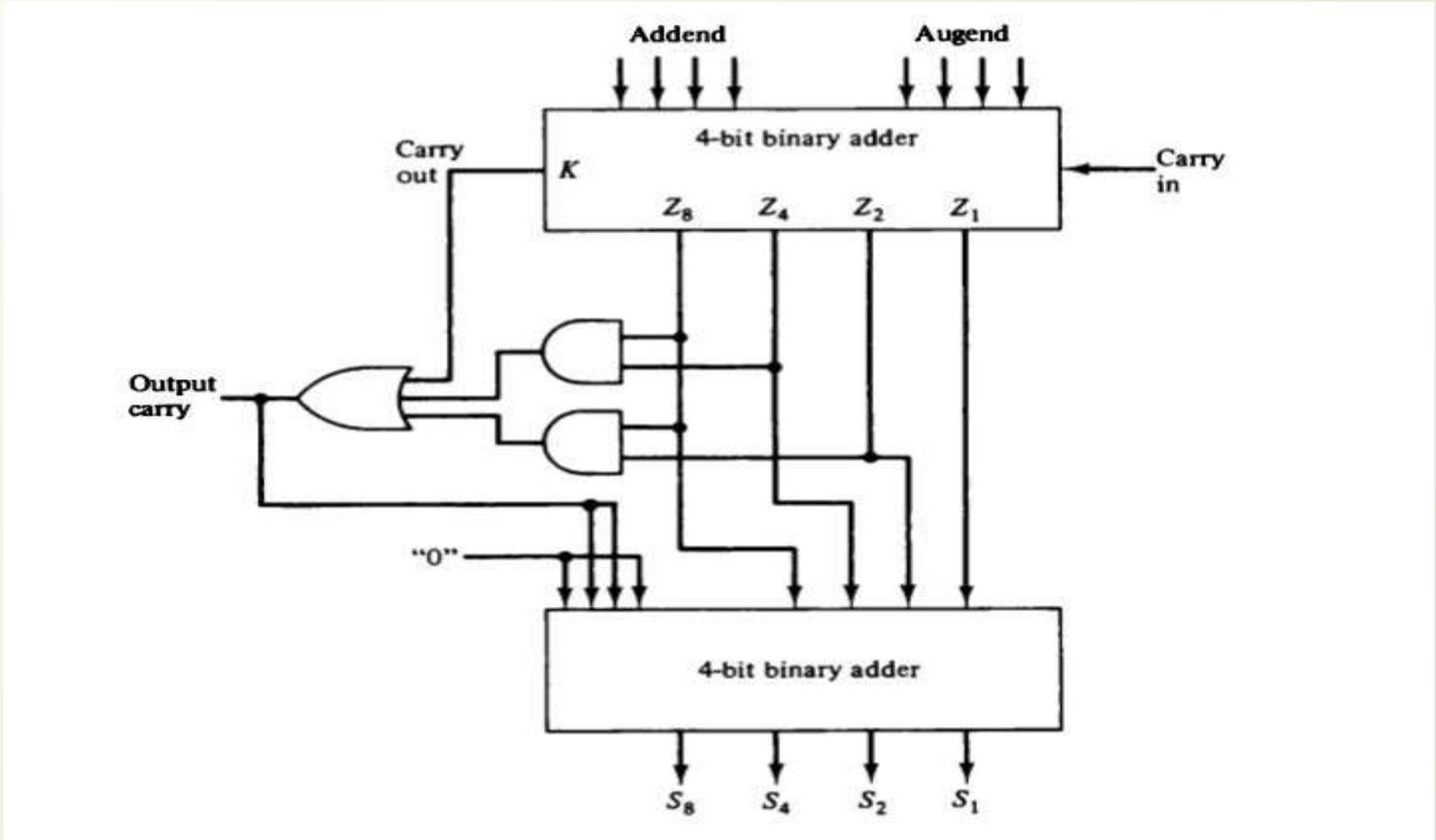
(When two decimal numbers are added, the maximum result is 19 => (9 + 9 +1(carry)))

Binary Sum					BCD Sum					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Contd...

(The condition for the correction and setting c=1 is:

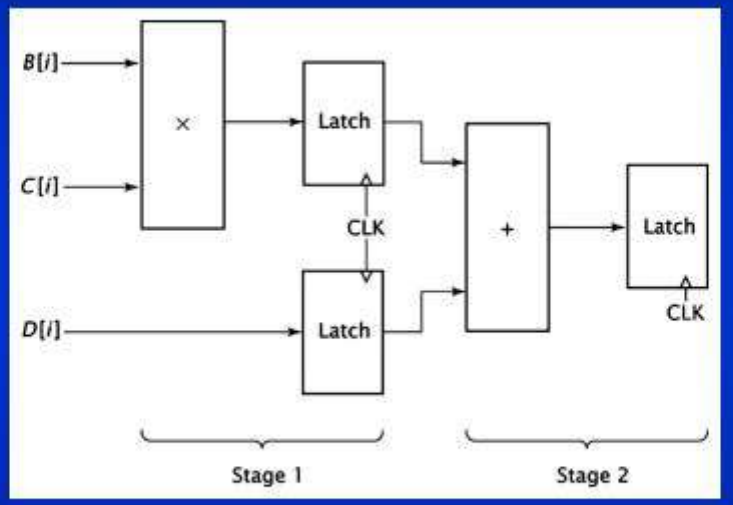
$c = k + z_8z_4 + z_8z_2 = 1$)



Specialized Arithmetic hardware

1. **PIPELINING**

FOR i = 1 TO 100 DO $\{A[i] \leftarrow (B[i] \cdot C[i]) + D[i]\}$



$$S_{100} = \frac{nT_1}{(k + n - 1)T_k} = \frac{100 \cdot 20\ \text{ns}}{(2 + 100 - 1) \cdot 10\ \text{ns}} = 1.98$$

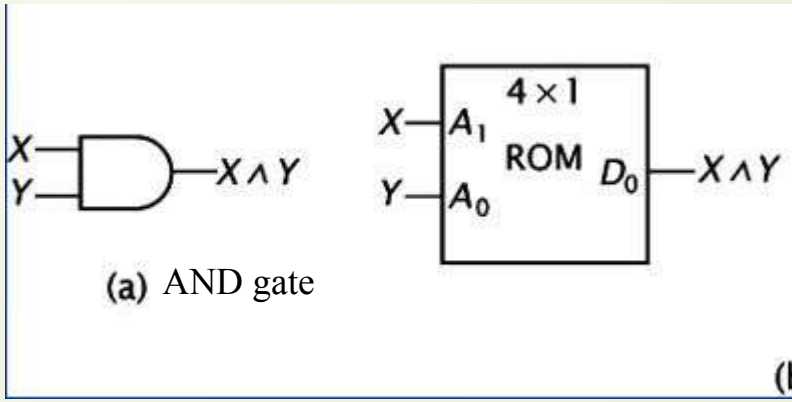
T_1 = time taken by non-pipelined unit to calculate 1 result. Requires $n.T_1$ time to calculate n results.
Pipelined unit requires k time units, each of duration T_k , to move the first piece of data through the pipeline.
Because, additional data enters the pipeline during every cycle, it will output the remaining n-1 results during next n-1 cycles.
=> thus, the pipelined unit requires (k + n – 1) cycles, each of T_k time, to calculate same results.
The speedup can be calculated by the formula above.

Contd.....

- Data enters into a stage of the pipeline, which performs some arithmetic operations on those data.
- The results are then passed to next stage which performs its operation and so on until the final computation ha been performed.
- Each stage performs only its specific function, no need to be capable of performing the task of any other stage.
- A pipeline does not speed up an individual computation. It improves the performance by overlapping computations where each stage can operate on different data simultaneously.
- The net effect is that output appears more quickly than in non-pipeline arithmetic unit, thereby increasing the throughput.

2. Lookup Tables:

- Any combinational circuit can be implemented by ROM is a lookup table.
- Inputs to the combinational circuit serve as the address input of the ROM.
- Data outputs of the ROM corresponds to the output of the combinational circuit.
- ROM is programmed with data such that the correct values are output for any possible input values.
- Consider a 4 x 1 ROM programmed to mimic a two input AND gate.

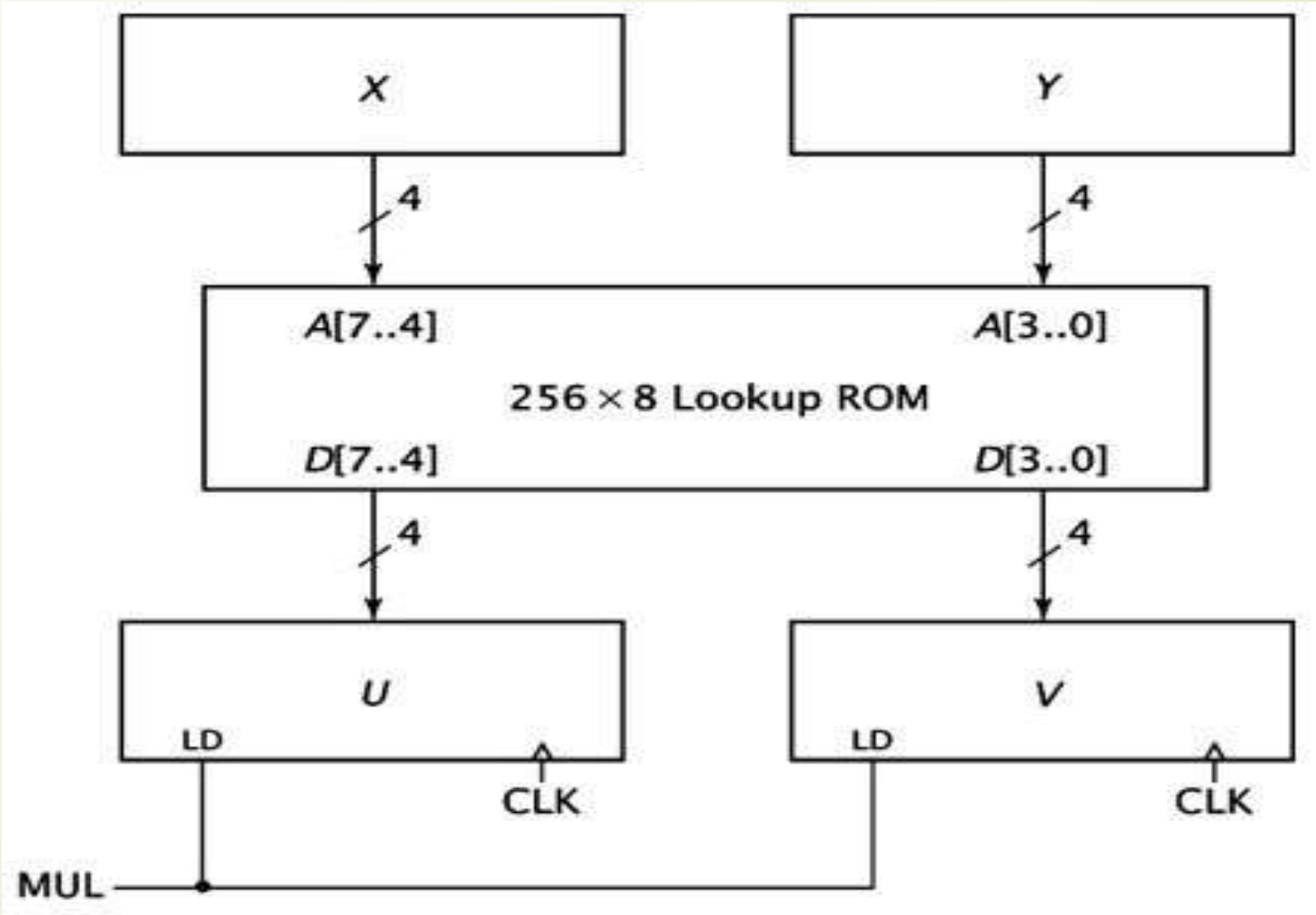


Address		Data
0	0	0
0	1	0
1	0	0
1	1	1

(b) Lookup ROM equivalent

- By programming ROM with data shown, it outputs the same values as the AND gate for all possible values of X and Y.

Contd..(a multiplier implemented using a lookup ROM)

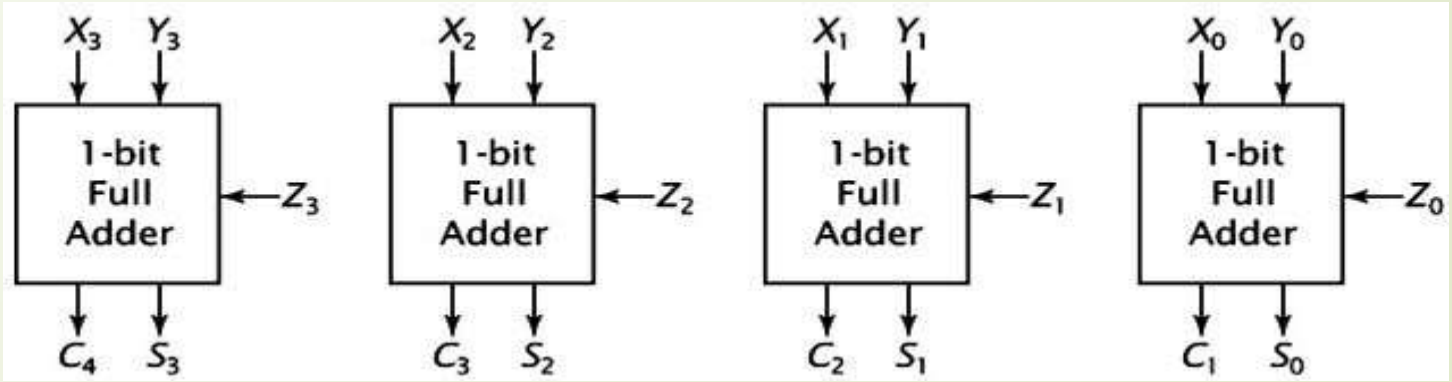


Contd...

- Registers X and Y supply the address inputs to lookup ROM.
- It's outputs are the product of X and Y, and are routed to registers U and V.
- Each of 256 locations must contain the 8-bit product of X and Y.
- For example, location 1011 1101 contains the data 1000 1111, 143, the product of 1011(11) and 1101(13).
- Advantages:
  => the hardware may be less complex than that of the original shift-add implementation.
  => can multiply numbers more quickly than the shift-add hardware.
  => However, size of the lookup ROM grows rapidly as the size of the operands increases.

3. Wallace Tree

- Combinational circuit used to multiply two numbers.
- Instead of performing additions using standard parallel adder, Wallace tree uses carry save adder and only one parallel adder.
- Carry save adder can add 3 values simultaneously. However, it does not output a single result, but provides both sum and a set of carry bits as output.



- To form the final sum, S and C must be added together because carry bits do not propagate through the adder.

- To use a carry save adder to perform multiplication, we first calculate the partial product of the multiplication and input them to carry save adder.

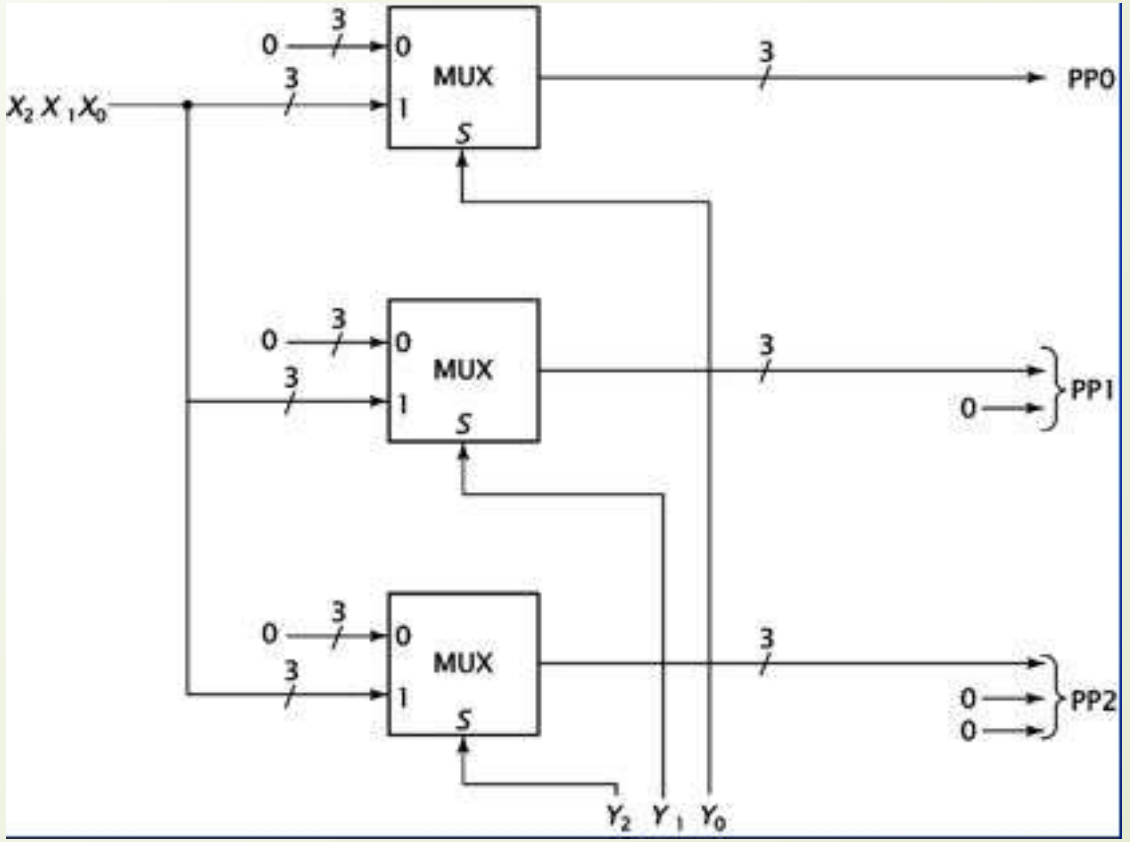
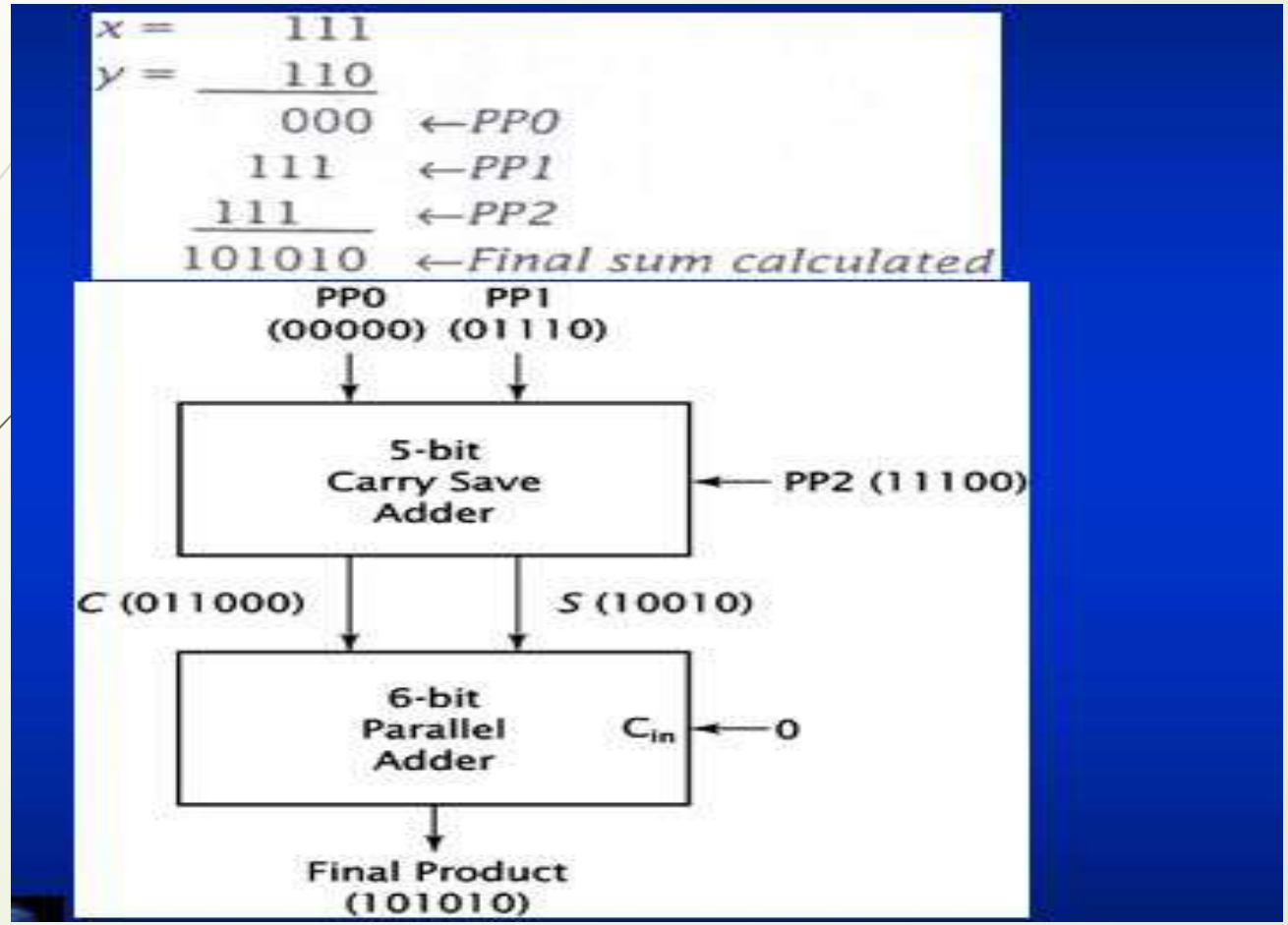


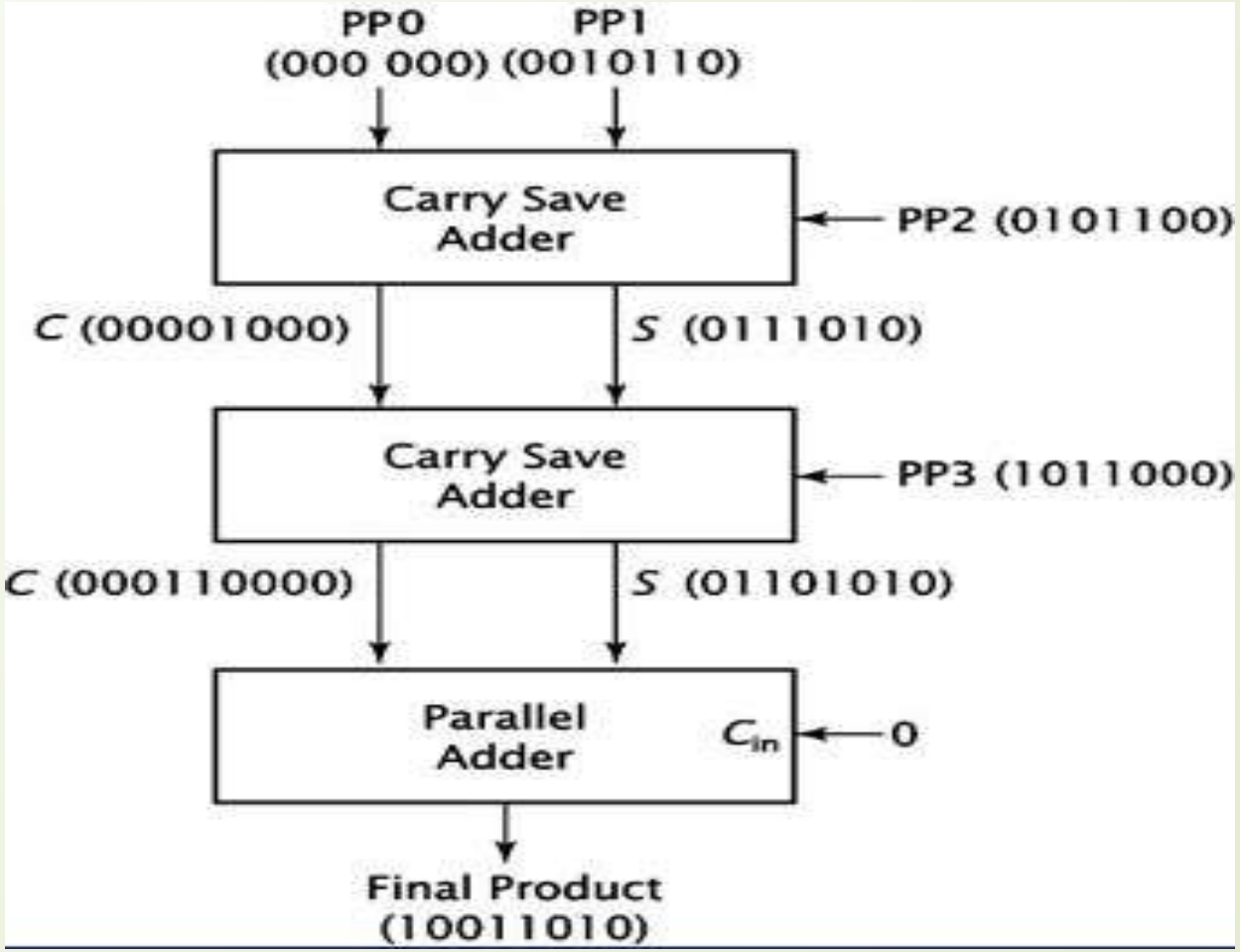
Fig: Generating partial products for multiplication using Wallace tree.

A 3 x 3 multiplier constructed using carry save adder.



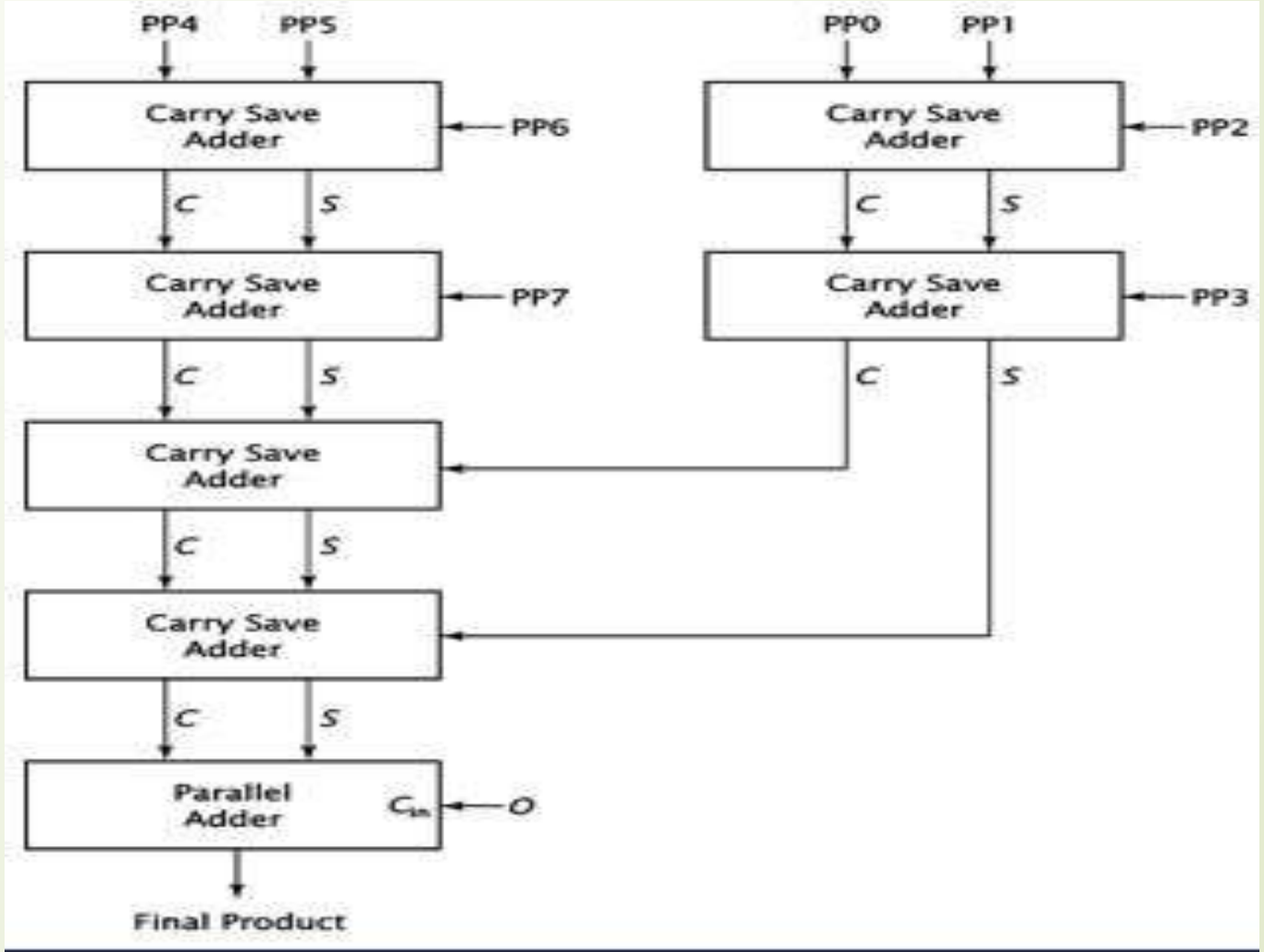
Contd..

A 4 x 4 Wallace tree multiplier.



Contd...

An 8 x 8 Wallace tree multiplier.



Floating point numbers

- Floating point format is similar to scientific notation.
- Using scientific notation, we can express a number in different ways.
  i.e. $-1234.5678 = -1.2345678 \times 10^{3}$
  $= -1234567.8 \times 10^{-3}$
- Computers can be efficient if each number can have only one representation (unique).
- So, floating point numbers must be normalized i.e. each number significand is a fraction with no leading zeros.
- Thus, the only valid floating point representation for this number is
  $-.12345678 \times 10^{4}$
  The normalized representation works well for every possible numbers except 0
- A computer stores floating point numbers in a pre-defined format.
  => Each number requires signed bit.
  => A significand of some pre-defined length.
  => An exponent of some given length.
  i.e. $X_S X_F X_E$ where $X_S=1$, $X_F=12345678$ and $X_E=4$

Contd...

- To represent negative exponents, one possibility is to use two's complement values, the prevalent practice is to use biasing.
- Assume that X_E has four bits, and should be able to represent all exponents from -8 to +7.
- To do so, a set bias value is added to the actual exponent and result is stored in $X_{E.}$
- -8 is represented as -8 + bias = -8 +8 = 0(0000).
- +7 is represented as +7 + bias = +7 +8 = 15(1111).

IEEE 754 floating point standard

- Defines set format and operation modes. Doesn't specify arithmetic procedures and hardware to be used.
- IEEE 754 standard is used in virtually all CPUs that have floating point capability.
- IEEE 754 standard specifies two precisions for floating point numbers.
  1. single precision number having 32 bits
  2. double precision number having 64 bits
- In single precision,
  1 bit for sign, 8 bit for exponent & 23 bit for significand
- In double precision,
  1 bit for sign, 11 bit for exponent & 52 bit for significand
- Significand falls in the range $1 \leq \text{significand} < 2$.
- Exponent uses a bias of 127, having range -126 t0 127.
- Exponent values 0000 0000(-127) and 1111 1111(128) are used for special numbers.

Contd...

- +19.5 = 10011.1 or 1.00111×2^4 in binary.
  - Not included, presence is implicit in this standard.
- For +19.5
  sign = 0
  significand = 001 1100 0000 0000 0000 0000
  exponent = 1000 0011 (131 = 4 + 127)
- For 0
  sign = 0
  significand = 000 0000 0000 0000 0000 0000
  exponent = 0000 0000

Addition and subtraction in Signed Notation

Addition and subtraction of signed-magnitude numbers

Operation	X_s	Y_s	AS	PM	X = 3, Y = 5	X = 5, Y = 3
(+X) + (+Y)	0	0	0	0	(+3) + (+5) = +8	(+5) + (+3) = +8
(+X) − (+Y)	0	0	1	1	(+3) − (+5) = −2	(+5) − (+3) = +2
(+X) + (−Y)	0	1	0	1	(+3) + (−5) = −2	(+5) + (−3) = +2
(+X) − (−Y)	0	1	1	0	(+3) − (−5) = +8	(+5) − (−3) = +8
(−X) + (+Y)	1	0	0	1	(−3) + (+5) = +2	(−5) + (+3) = −2
(−X) − (+Y)	1	0	1	0	(−3) − (+5) = −8	(−5) − (+3) = −8
(−X) + (−Y)	1	1	0	0	(−3) + (−5) = −8	(−5) + (−3) = −8
(−X) − (−Y)	1	1	1	1	(−3) − (−5) = +2	(−5) − (−3) = −2

$PM'1$: $U_s \leftarrow X_s$, $CU \leftarrow X + Y$

$PM1$: $CU \leftarrow X + Y' + 1$, $OVERFLOW \leftarrow 0$

$PM'2$: $OVERFLOW \leftarrow C$

$CZ'PM2$: $U_s \leftarrow X_s$

$CZPM2$: $U_s \leftarrow 0$

$C'PM2$: $U_s \leftarrow X_s'$, $U \leftarrow U' + 1$

2: $FINISH \leftarrow 1$

Contd...

Examples of addition and subtraction of signed-magnitude numbers

$U_sU = X_sX + Y_sY$
$X_sX = +3 = 0\ 0011$
$Y_sY = +5 = 0\ 0101$
PM'1: $U_s \leftarrow 0$, $CU \leftarrow 0\ 1000$
PM'2: $OVERFLOW \leftarrow 0$
Result: $U_sU = 0\ 1000 = +8$

(a)

$U_sU = X_sX - Y_sY$
$X_sX = +5 = 0\ 0101$
$Y_sY = +5 = 0\ 0101$
PM1: $CU \leftarrow 1\ 0000$, $OVERFLOW \leftarrow 0$
CZPM2: $U_s \leftarrow 0$
Result: $U_sU = 0\ 0000 = +0$

(c)

$U_sU = X_sX + Y_sY$
$X_sX = +13 = 0\ 1101$
$Y_sY = +5 = 0\ 0101$
PM'1: $U_s \leftarrow 0$, $CU \leftarrow 1\ 0010$
PM'2: $OVERFLOW \leftarrow 1$
Result: *OVERFLOW*

(e)

$U_sU = X_sX + Y_sY$
$X_sX = +5 = 0\ 0101$
$Y_sY = -3 = 1\ 0011$
PM1: $CU \leftarrow 1\ 0010$, $OVERFLOW \leftarrow 0$
CZ'PM2: $U_s \leftarrow 0$
Result: $U_sU = 0\ 0010 = +2$

(b)

$U_sU = X_sX - Y_sY$
$X_sX = +3 = 0\ 0011$
$Y_sY = +5 = 0\ 0101$
PM1: $CU \leftarrow 0\ 1110$, $OVERFLOW \leftarrow 0$
C'PM2: $U_s \leftarrow 1$, $U \leftarrow 0010$
Result: $U_sU = 1\ 0010 = -2$

(d)

$U_sU = X_sX - Y_sY$
$X_sX = -7 = 1\ 0111$
$Y_sY = +13 = 0\ 1101$
PM'1: $U_s \leftarrow 1$, $CU \leftarrow 1\ 0100$
PM'2: $OVERFLOW \leftarrow 1$
Result: *OVERFLOW*

(f)