Design of Arithmetic circuits

5/8/14

(1)

Adders

Fast addition of 2-n bit unsigned numbers

we 2-level combination ckt expressed in

SOP or POS of n-I/p variables

$C(n)$ — # gates

$f(n)$ — fan-in

Practical circuits < Multilevel combinational / Sequential ckts

Half adder

$S_i = x_i \oplus y_i$

$C_i = x_i y_i$

Full adder

$S_i = x_i \oplus y_i \oplus c_{i-1}$

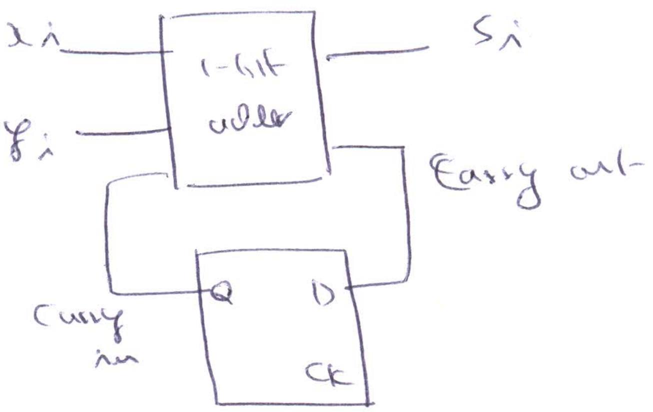
$C_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1}$

x_i	y_i	c_{i-1}	S_i	c_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Serial adder

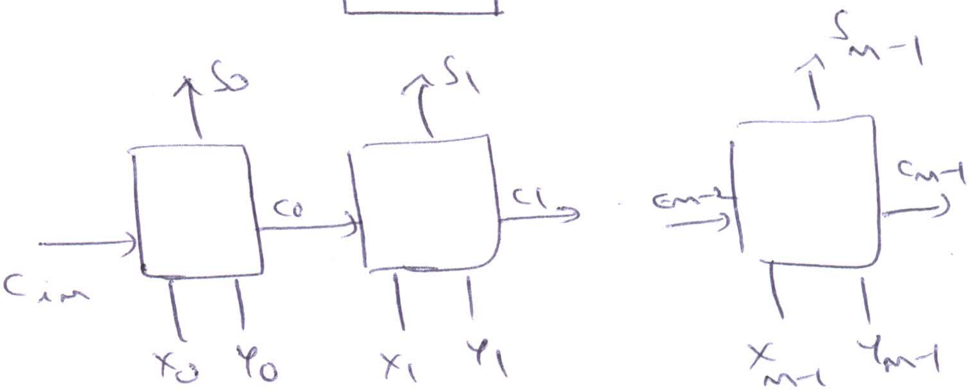
For n-bit addition

n - clock cycles



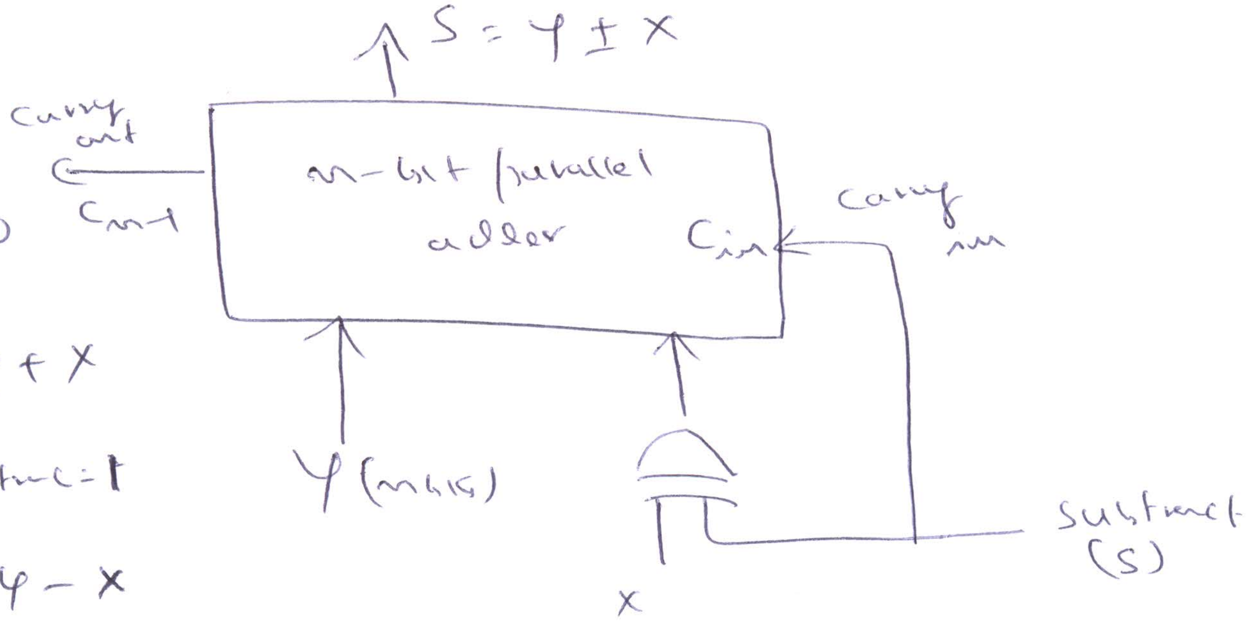
Parallel adder

n-bit ripple carry adder



Time delay = $\underline{nd}$ → delay in FA stage

n-bit 2's complement adder - subtractor



sub when $S = 0$

$S = Y + X$

when subtract = 1

$S = Y - X$

Binary Subtracter

S_i = difference bit

b_{i-1} = borrow-in

b_i = borrow out

$$S_i = x_i \oplus y_i \oplus b_{i-1}$$

$$b_i = x_i \bar{y}_i + \bar{x}_i b_{i-1} + \bar{y}_i b_{i-1}$$

over flow

unsigned number $\Rightarrow$ $c_{n-1} = 1$

Signed Numbers: both +ve $\Rightarrow$ $c_{n-2} = 1$; both −ve $\Rightarrow$ $c_{n-2} = 0$

$$V = x_{n-1} y_{n-1} \bar{c}_{n-2} + \bar{x}_{n-1} \bar{y}_{n-1} c_{n-2}$$

$$c_{n-1} = x_{n-1} y_{n-1} + x_{n-1} c_{n-2} + y_{n-1} c_{n-2}$$

$$\Rightarrow V = c_{n-1} \oplus c_{n-2}$$

High Speed Adder

Delay = Sum of logic gate delays along the longest signal propagation path through N/W

n-bit ripple adder

$c_{n-1} \rightarrow 2(n-1)$ gate delays

$s_{n-1} \rightarrow 2(n-1) + 1$ XOR delay

$c_n \rightarrow 2n$ gate delays

overflow $\rightarrow 2n+2$ gate delays

Reducing delay

1) Faster Electronic technology

2) modify the logic gate structure (Fast propagation of carry)

Carry - lookahead - addition

$$S_i = x_i \oplus y_i \oplus c_i$$

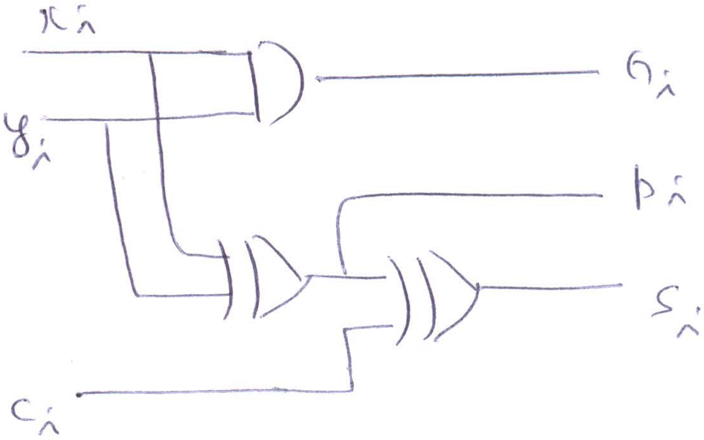
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$= x_i y_i + c_i(x_i + y_i)$$

$$= G_i + c_i P_i \qquad \Bigg| \quad G_i = x_i y_i, \quad P_i = x_i + y_i$$

G -> generate fn.

P => propagate fn



$$c_{i+1} = G_i + P_i\left(G_{i-1} + c_{i-1} P_{i-1}\right)$$

$$= G_i + P_i G_{i-1} + P_i P_{i-1} c_{i-1}$$

$$= G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \dots + P_i P_{i-1} \cdots P_1 G_0$$

$$+ P_i P_{i-1} \cdots P_0 c_0$$

1 gate delay => G_i & P_i

2 gate delays => c_{i+1} (AND-OR)

1 gate delay => S_i (X-OR gate)

=> 4-gate delays

overflow

unsigned addition

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

$c_n = 1$

$s < a$

$s < b$

unsigned subtract

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

$C_n = 0$

$a < s$

$a < b$

Multiplier

Simple Method

A	0	1	1
B	1	0	1

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

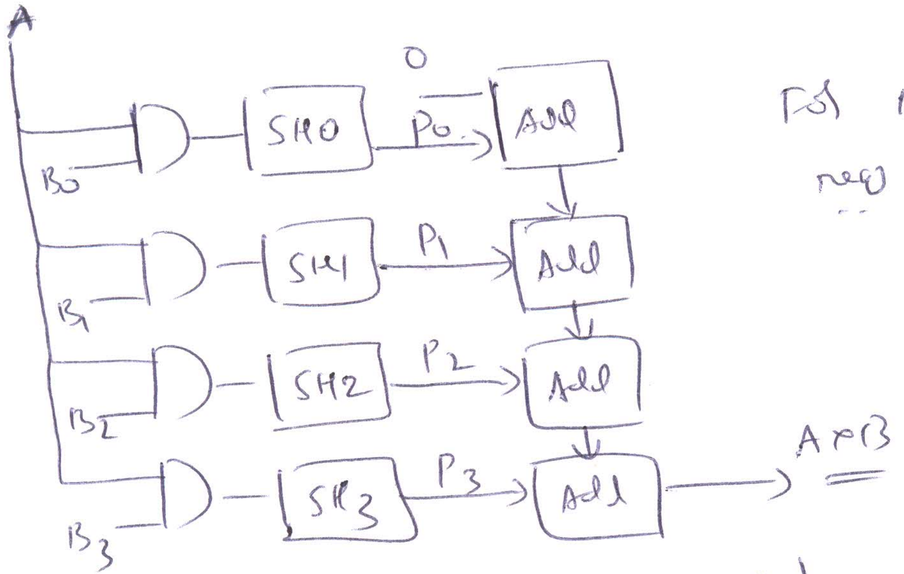
$\Rightarrow \sum_{i=0}^{n-1} A \cdot B_i \, 2^i$

$P_0 \rightarrow$ 0 0 0 0 0

$P_1 \rightarrow$ 0 0 0 1 1

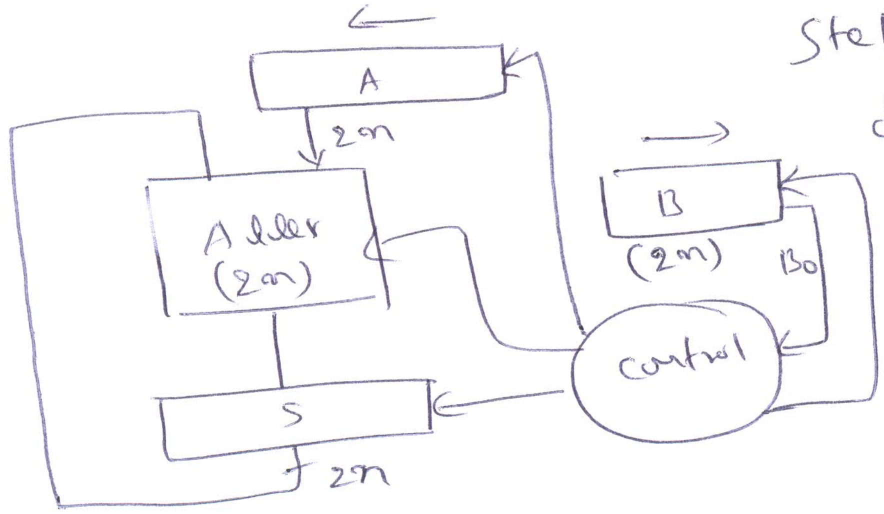
$P_2 \rightarrow$ 0 0 0 1 1

$P_3 \rightarrow$ 0 1 1 1 1



For N×N multiplier need N adders.

Shift add Multiplier (sequential)



Step1; $i=0$; $s=0$

do {

Step2:

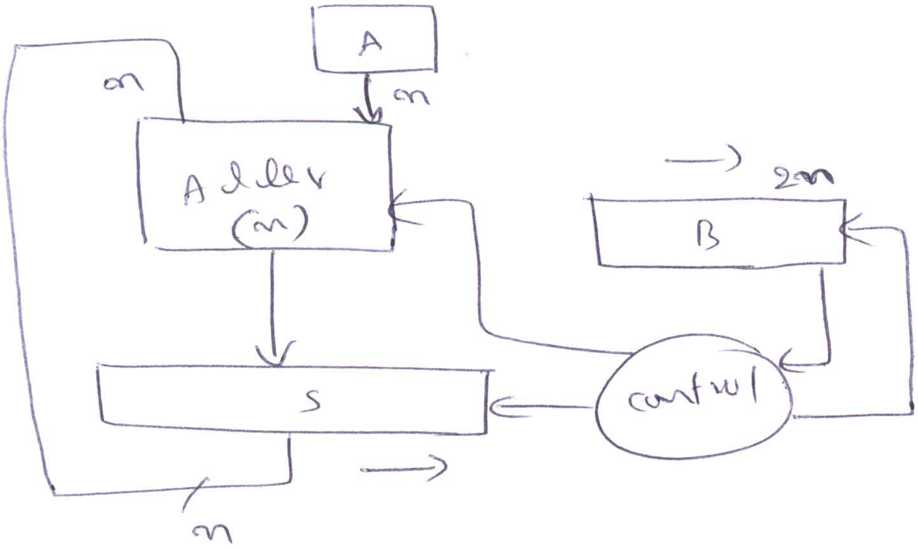
if (B_0) S += A

A = 2×A

B = B/2; i++

} while (i<n)

Sequential shift add multiplier 2



Step 1: $i=0$, $S=0$

do

Step 2:

if (B_0) S_H += A

$S = S/2$

$B = B/2$

$i = i+1$

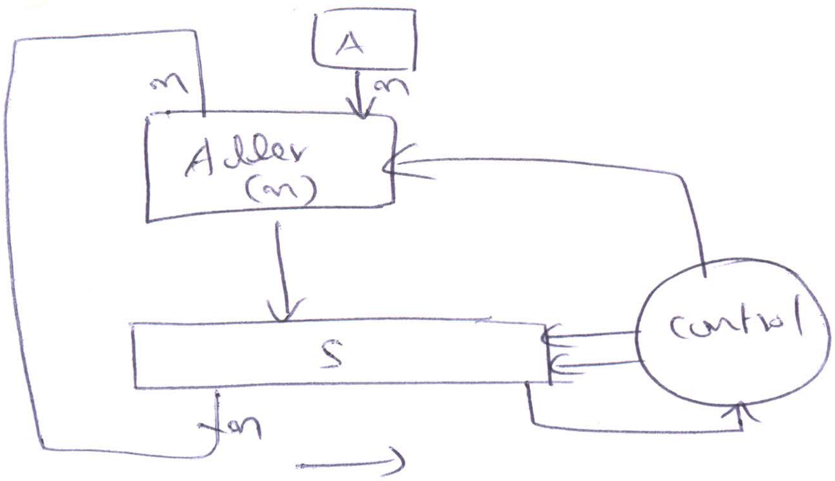
}

while $i < m$

✓ Reducing size of A from 2m to m

✓ Reducing the size of adder from 2m to m

Sequential shift add multiplier 3



Step 1: $i=0$; $S = \boxed{0|B}$

do

{

Step 2

if (S_0) S_H += A

$S = S/2$; i++

}

while ($i < m$)

✓ Avoid the use of separate Reg for B

H, L Reg pair for holding 64 bits

Signed Multiplication

Common expression to rep both positive as well as negative values.

$$B = -B_{n-1}2^{n-1} + \sum_{i=0}^{n-2} B_i 2^i$$

for $B \geq 0$

$$B = \sum_{i=0}^{n-1} B_i 2^i = -B_{n-1}2^{n-1} + \sum_{i=0}^{n-2} B_i 2^i$$

$(\because B_{n-1} = 0)$

for $B < 0$, $B = -|B|$

$$|B| = 2^n - \sum_{i=0}^{n-1} B_i 2^i = 2^n - B_{n-1}2^{n-1} - \sum_{i=0}^{n-2} B_i 2^i$$

$$= B_{n-1}2^{n-1} - \sum_{i=0}^{n-2} B_i 2^i$$

$$B = -|B| = -B_{n-1}2^{n-1} + \sum_{i=0}^{n-2} B_i 2^i$$

$$= \cancel{-B_{n-1}2^{n-1} + B_{n-2}\left(2^{n-1} - 2^{n-2}\right) + \cdots + B_0}$$

$$= -B_{n-1}2^{n-1} + B_{n-2}2^{n-2} + \cdots + B_1 2 + B_0$$

$$= -B_{n-1}2^{n-1} + B_{n-2}2^{n-1} - B_{n-2}2^{n-2} + \cdots + B_1 2^2 - B_1 2 + B_0 2 - B_0 2^0$$

$$= \sum_{i=0}^{n-1} \left(B_{i-1} - B_i\right) 2^i \quad \text{where } B_{-1} = 0$$

$$\therefore A \times B = \sum_{i=0}^{n-1} A \cdot \left(B_{i-1} - B_i\right) 2^i$$

Comparison bet signed ~~and~~ and unsigned multiply

unsigned		signed		
B_i	operation	B_i	B_{i-1}	operation
0	no addition	0	0	no addition
1	add A	0	1	add A
		1	0	sub A
		1	1	no addition

Motivation of Booth's Algorithm

$0\,0\,\underbrace{0\,1}_{\text{add}}\,1\,1\,1\,1\,1\,1\,\underbrace{1\,0}_{\text{sub}}\,0\,0\,0$

additions will come down

Range of Multiplication o/p

n-bit unsigned multiplication $0 \le \text{result} \le (2^n - 1)^2$

$\le 2^{2n} - 2\cdot 2^n + 1$

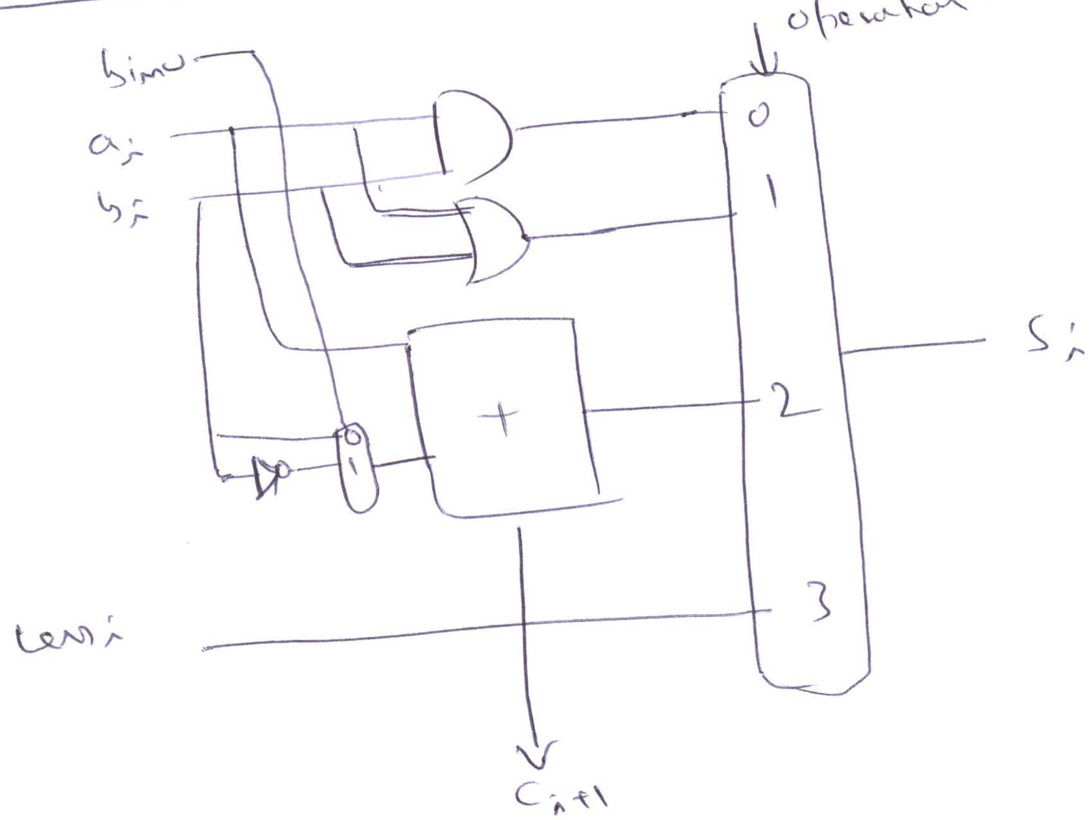
$\underbrace{1\,1 - - - - 1}_{n-2}\;\underbrace{0\,0 - - 0}_{n}\;\underbrace{1}_{1}$

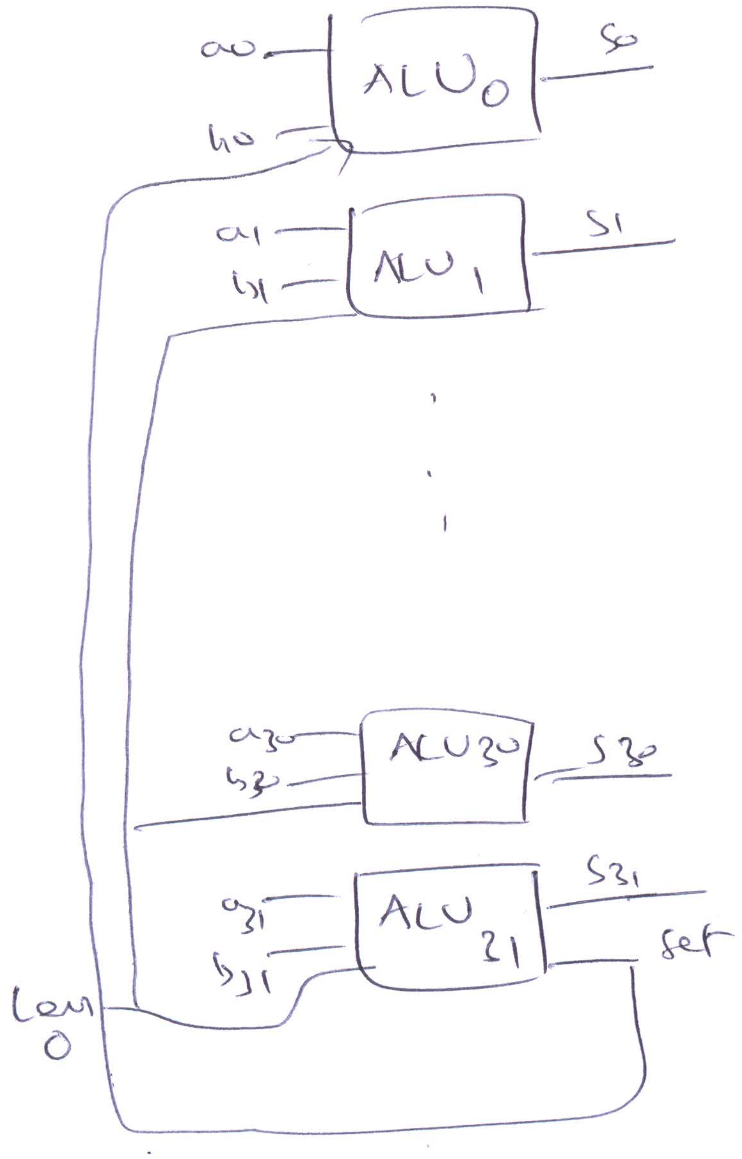
Sign multiplication

$-2^{n-1} - 2^{2n-2} \le \text{result} \le 2^{2n-2}$

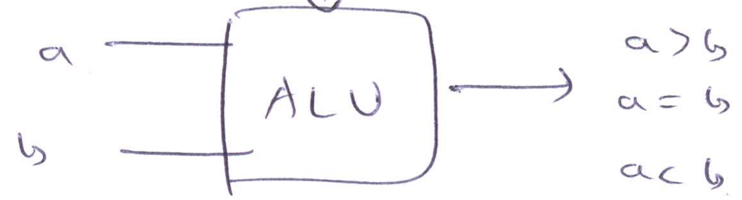
No overflow

ALU for Add, Sub, AND, OR, SLT





Comparison

① 

② Direct comparison

greater than

From LSB

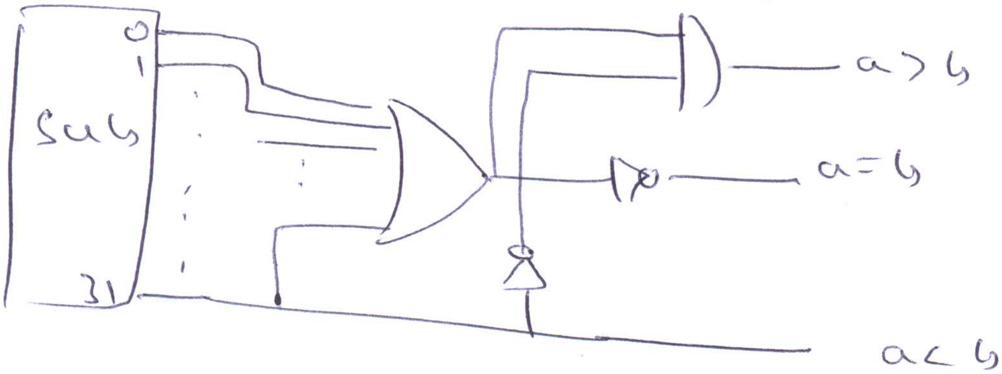
$$a_{0 \cdots i+1} > b_{0 \cdots i+1} = (a_{0 \cdots i} > b_{0 \cdots i})(a_{i+1} + \overline{b_{i+1}}) + (a_{0 \cdots i} = b_{0 \cdots i})\, a_{i+1} \overline{b_{i+1}}$$

From MSB

$$a_{i-1 \cdots 31} > b_{i-1 \cdots 31} = (a_{i \cdots 31} > b_{i \cdots 31}) + (a_{i \cdots 31} = b_{i \cdots 31})\, a_{i-1} \overline{b_{i-1}}$$

Equal

$$a = b \Rightarrow \prod_{i=0}^{31} (a_i \overline{\oplus} b_i)$$



Division

$$A \div B = Q + R/B$$

$$A = Q \times B + R$$

```
0100 ) 00001101 ( 00011
       01000000            —— 2^4 × B
       --------
       00001101
        0100 - -           —— 2^3 × B
       --------
       00001101
         0100 - - -        —— 2^2 × B
       --------
       00001101
          0100 . .         —— 2^1 × B
       --------
            0101
            0100           —— 2^0 × B
       --------
            (1)
```

Step 1 $i = 0$, $R = A$, $Q = 0$
$D = B$

do {

Step 2

if $(D \times 2^{n-i-1} \le R)$ then

$R = R - D \times 2^{n-i-1}$

$Q_{n-i-1} = 1$

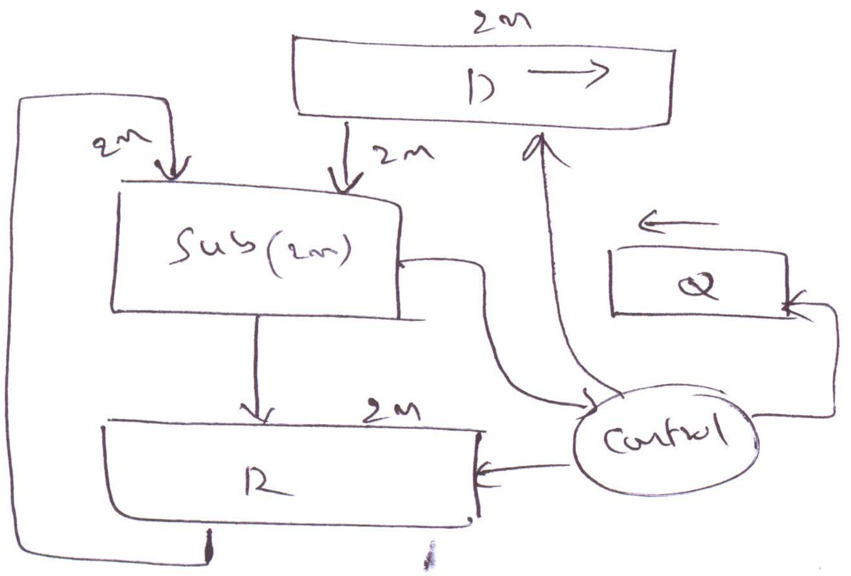
else

$Q_{n-i-1} = 0$

} while $i < n$

Loop invariant $A = Q \times D + R$ & $0 \le R < D \times 2^{n-i}$

Design-1



Step-1: $i = 0$, $R = A$, $Q = 0$, $D = B \times 2^{n-1}$

do {

Step 2:

if $(D \le R)$

then $R = R - D$; $Q = 2Q + 1$

else

$Q = 2Q$

$D = D/2$; i++

} while $(i < n)$

Reducing Substractor Size

$A = Q \times B + R$

Step 1: $i = 0$; $R = 2 \times A$; $Q = 0$; $D = B$

do {

Step 2:

if ($D \le RH$) $RH = RH - D$; $Q = 2Q + 1$

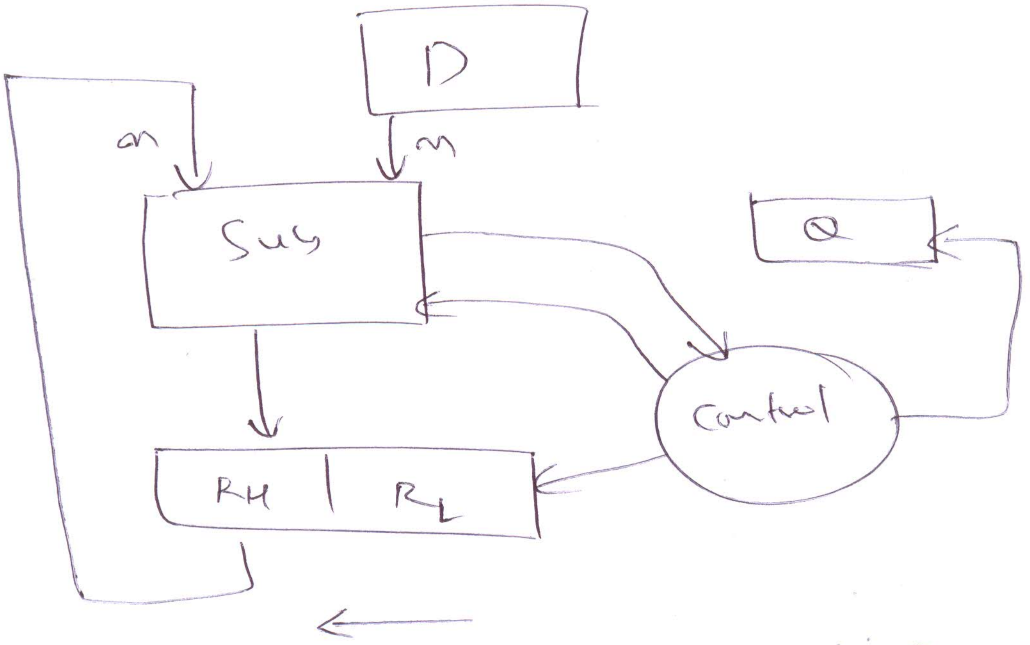
else

$Q = 2Q$

$R = 2R$; $i{+}{+}$

} while ($i < n$)

Design-2



Reducing Registers

$A = Q \times B + R$

Step 1: $i = 0$; $R = 2 \times A$; $D = B$

do Step 2: if ($D \le RH$) $RH = RH - D$; $R = 2R + 1$

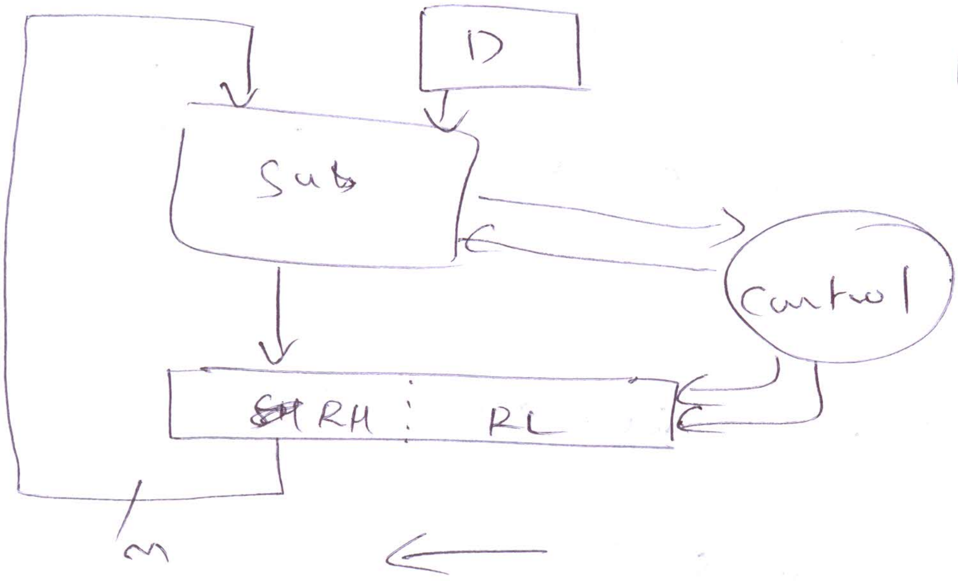
else

$R = 2R$

} $i{+}{+}$

while ($i < n$)

Design-3



Restoring Division

Non-restoring approach

if $(D \le R)$ $R = R - D$; $D = D/2$; $Q = 2Q+1$

Else $D = D/2$ $Q = 2Q$

i++

Restored approach

$R = R - D$

if $(R<0)$ $R = R + D$; $Q = 2Q$

Else $Q = 2Q+1$

$D = D/2$; i++

Delaying Restoration

$R = R + D$ & $R = R - D/2$ $\Rightarrow$ $R = R + D/2$

Improved restoring division

Step 1: $i = 0$; $R = A$; $Q = 0$; $D = B \times 2^{m-1}$

do {

Step 2A:

if $(R < 0)$ $R = R + D$ else $R = R - D$

Step 2B:

if $(R < 0)$ $Q = 2Q$ else $Q = 2Q + 1$

$D = D/2$; i++

} while $i < m$

if $(R < 0)$ $R = R + B$

Further improvement

Step 1 $i = 0$; $R = A$; $Q = 0$; $D = B \times 2^{m-1}$

do {

Step 2 if $(R < 0)$

$R = R + D$; $Q = 2 \times Q - 1$

Else

$R = R - D$; $Q = 2 \times Q + 1$

$D = D/2$; i++

} while $(i < m)$

if $(R < 0)$ $R = R + B$; $Q = Q - 1$

Extending to Signed division

Step 1: $i = 0$ $R = A, Q = 0$; $D = B \times 2^{n-1}$

do {

Step 2

if ($R_{n-1} \neq D_{n-1}$)

$R = R + D$; $Q = 2Q - 1$

Else

$R = R - D$; $Q = 2Q + 1$

$D = D/2$; i++

} while ($i < n$)

if ($R_{n-1} \neq D_{n-1}$) $R = R + B$; $Q = Q - 1$

Signs in Signed division

Dividend	Divisor	Quotient	Remainder
+	+	+	+
+	−	−	+
−	+	−	−
−	−	+	−

Carry-lookahead adder

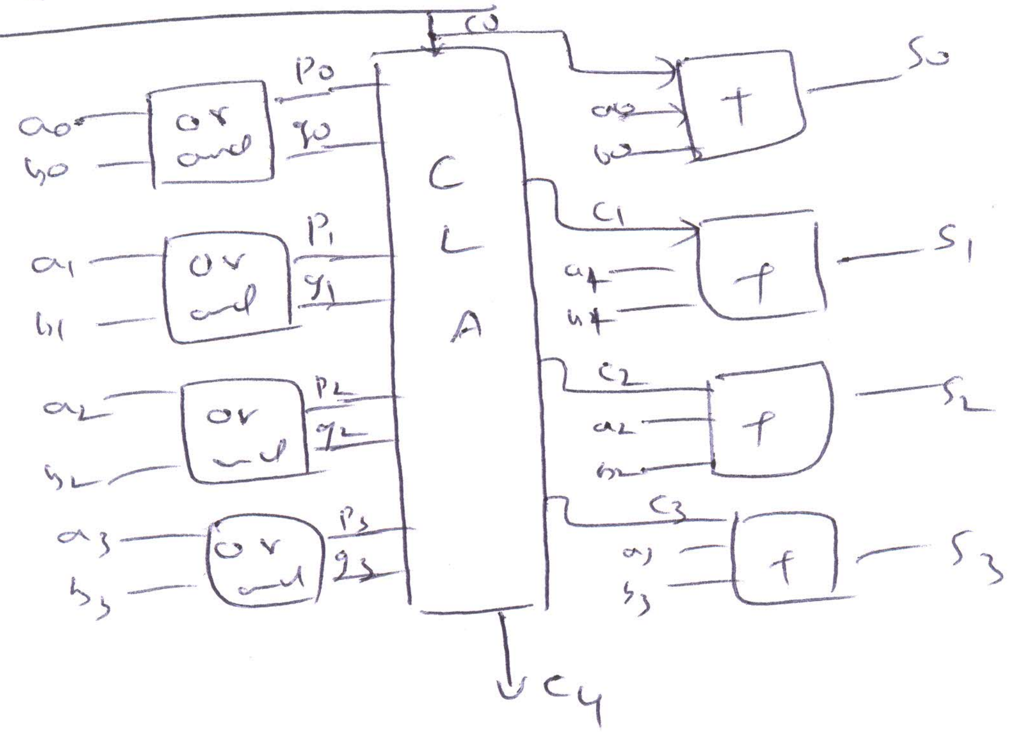
$$c_1 = p_0 c_0 + g_0$$

$$c_2 = p_1 c_1 + g_1 = p_1 p_0 c_0 + p_1 g_0 + g_1$$

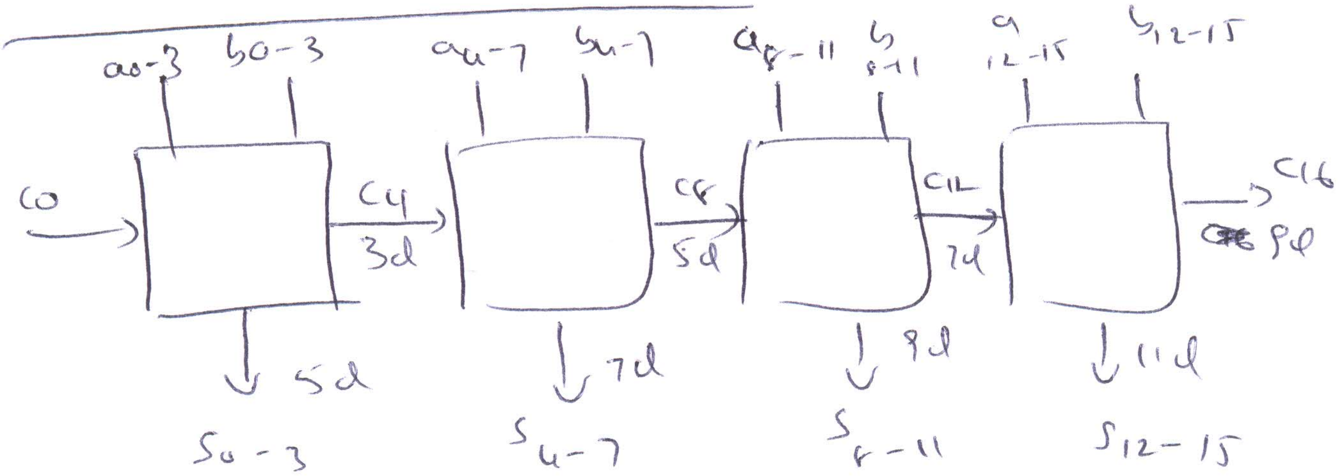
$$c_3 = p_2 c_2 + g_2 = p_2 p_1 p_0 c_0 + p_2 p_1 g_0 + p_2 g_1 + g_2$$

$$c_4 = p_3 c_3 + g_3 = p_3 p_2 p_1 p_0 c_0 + p_3 p_2 p_1 g_0 + p_3 p_2 g_1 + p_3 g_2 + g_3$$

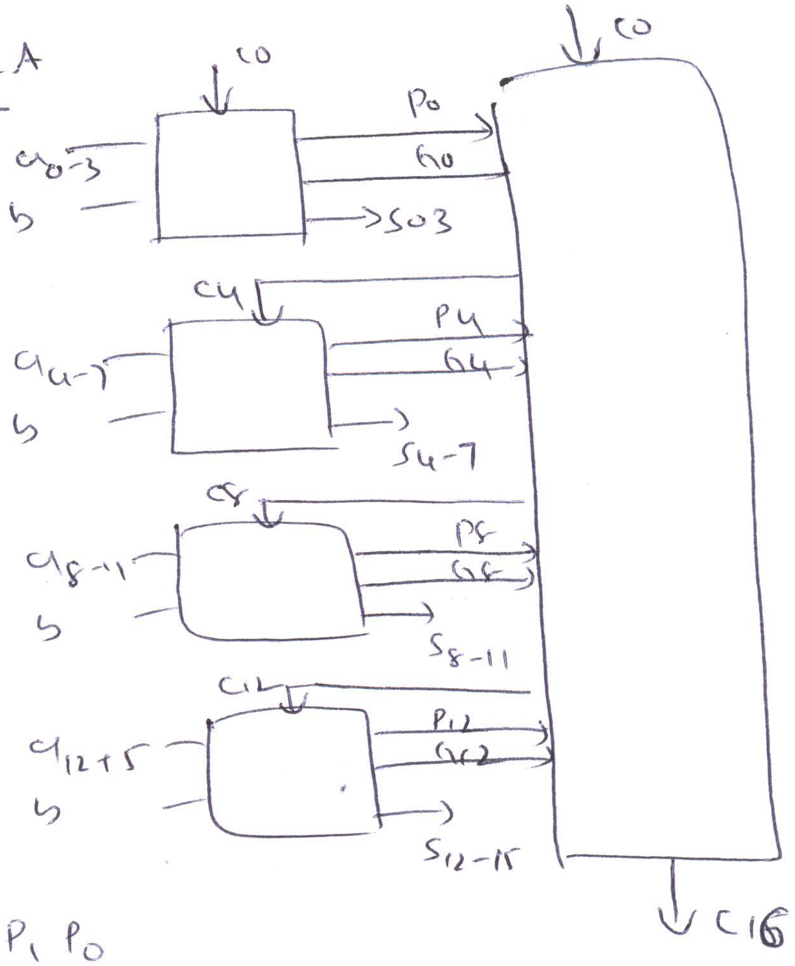
4-bit CLA adder



16-bit adder with 4-bit CLA)



2-Levels of CLA



$$P_0 = P_3 P_2 P_1 P_0$$

$$G_0 = P_3 P_2 P_1 g_0 + P_3 P_2 g_1 + P_3 g_2 + g_3$$

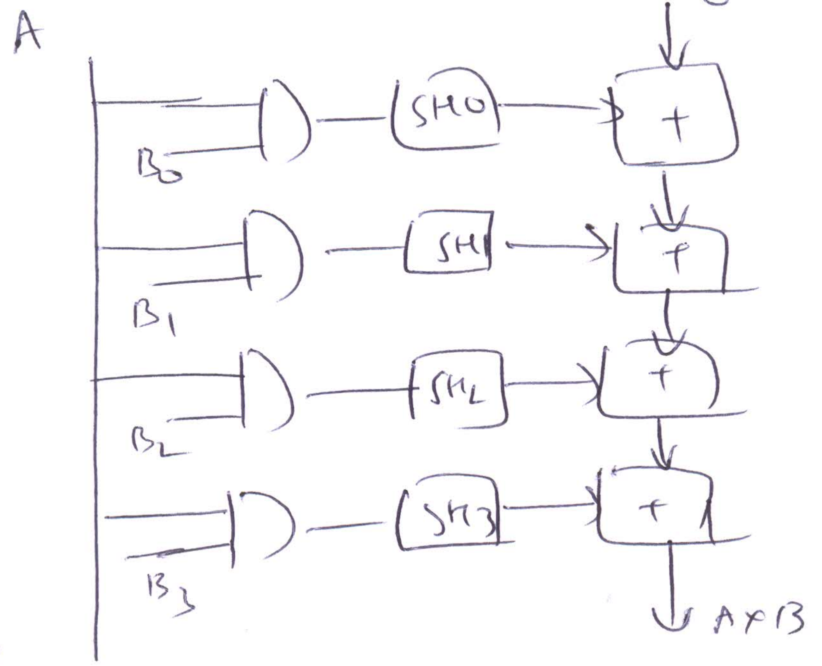
$$C_4 = P_0 C_0 + G_0$$

$$C_8 = P_4 P_0 C_0 + P_4 G_0 + G_4$$

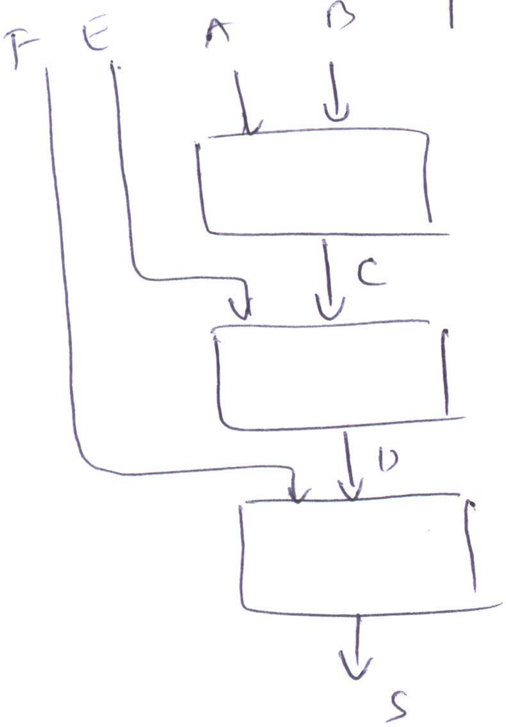
$$C_{12} = P_8 P_4 P_0 C_0 + P_8 P_4 G_0 + P_8 G_4 + G_8$$

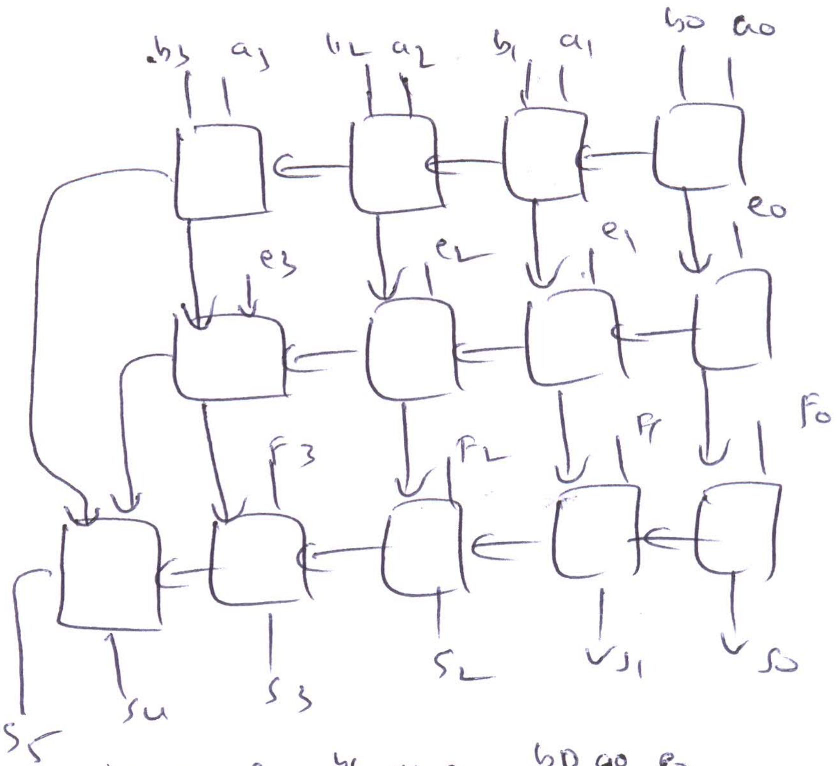
$$C_{16} = P_{12} P_8 P_4 P_0 C_0 + P_{12} P_8 P_4 G_0 + P_{12} P_8 G_4 + P_{12} G_8 + G_{12}$$

Array Multiplier (Carry Save adder)

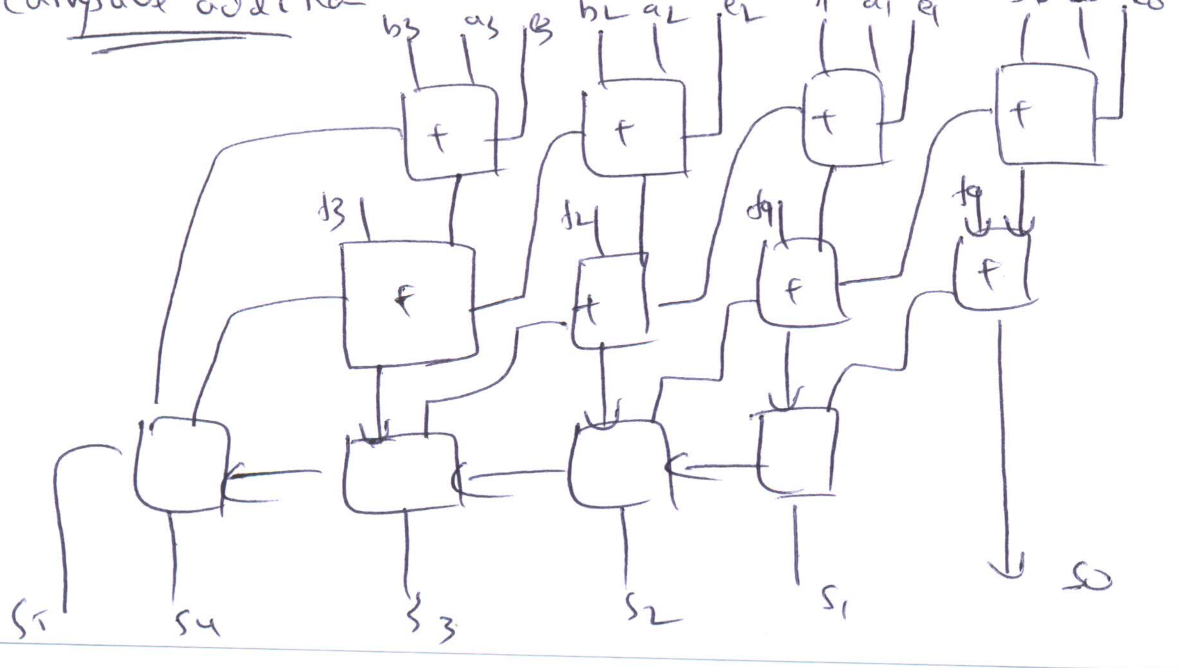


$$A \times B = \sum_{i=0}^{n-1} A \cdot B_i 2^i$$





Carry save addition



Floating Point Numbers & System Design

12/8/14

Datatypes: Integer, rational, real & complex

Scientific Applications < Astronomy (distance bet Sun & Pluto) 5.9×10^{12} m
weight of an electron 9.1×10^{-28} gms

Representing Fractions

Integer pairs (for rational numbers)

$\boxed{5}$ $\boxed{8}$ => 5/8

Strings with explicit decimal point

$\boxed{-}\boxed{2}\boxed{4}\boxed{7}\boxed{.}\boxed{0}\boxed{9}$ => −247.09

Fixed point rep (implicit pt in a fixed place)

0100110.01001

Floating point

Fraction × base^{power}

Precision & accuracy → Mantissa

Base & power → range

$0.6 = 0.1001\,1001\,1001\ldots$

$0.6 \times 2 = 1 + 0.2$

$0.2 \times 2 = 0 + 0.4$

$0.4 \times 2 = 0 + 0.8$

$0.8 \times 2 = 1 + 0.6$

$0.6 \times 2 = 1 + 0.2$

⋮

FP Number with base 2

$(-1)^S \times F \times 2^E$

S = Sign

F = Fraction / Mantissa / significant — Fixed point no

E = Exponent (± integer)

IEEE - 754

Single precision

1	8	23
S	E	F

Double precision

1	11	52
S	E	F

Rep of F in IEEE 754 (Representation of mantissa)

1	23

1	52

$1 \le F \le 2 - 2^{-23} \approx 2$ $\qquad$ $1 \le F \le 2 - 2^{-52} \approx 2$

Rep of exponent

- Single precision (8) : bias 127
- Double precision (11) : bias 1023

0 - 255 => 1 - 254 => 1-127 to 254-127

-126 to 127

Double precision

0 - 2047 => 1 - 2046 => 1-1023 to 2046-1023

-1022 to 1023

Rep of zero

All bits (S, F, E) are zeros.

Testing & comparing

zero, +ve, -ve is same as integers

magnitude comparison is same as integers

Overflow & underflow

Largest +ve & -ve Number (SP) = $\pm(2-2^{-23}) \times 2^{127} \approx \pm 2 \times 10^{38}$

" " " (DP) = $\pm(2-2^{-52}) \times 2^{1023} \approx \pm 2 \times 10^{308}$

Smallest +ve & -ve " (SP) = $\pm 1 \times 2^{-126} \approx \pm 2 \times 10^{-38}$

" " (DP) = $\pm 1 \times 10^{-1022} \approx \pm 2 \times 10^{-308}$

Precesion SP $\rightarrow \frac{23}{\log_2 10} \approx 8$ DP $\rightarrow \frac{52}{\log_2 10} \approx 17$

Floating Point operations

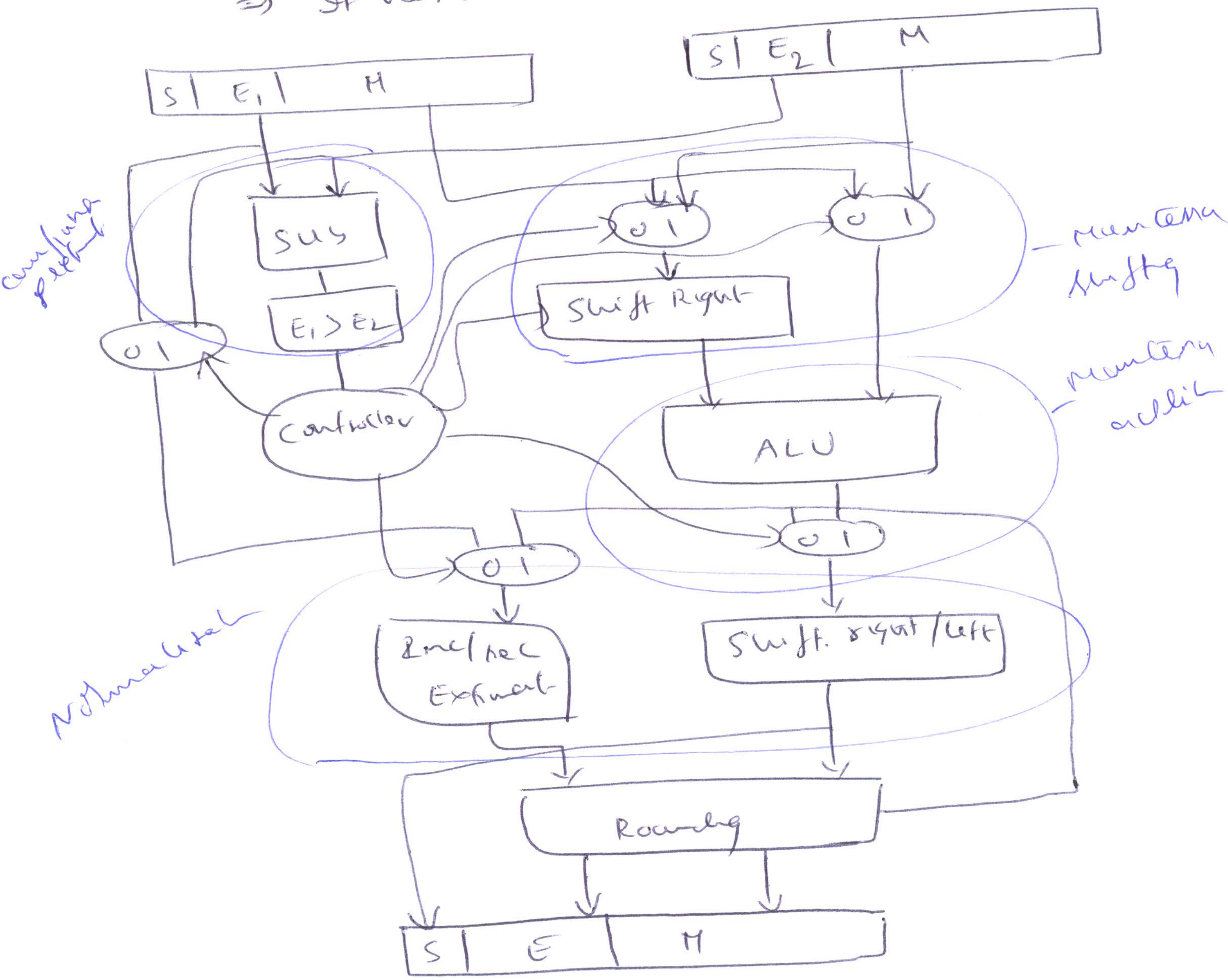
Add / Subtract $\left[(-1)^{S_1} \times F_1 \times 2^{E_1}\right] \pm \left[(-1)^{S_2} \times F_2 \times 2^{E_2}\right]$

if $E_1 > E_2$

$$\left[(-1)^{S_1} \times F_1 \times 2^{E_1}\right] \pm \left[(-1)^{S_2} \times F_2' \times 2^{E_1}\right] ; \quad F_2' = \frac{F_2}{2^{E_1 - E_2}}$$

$$\Rightarrow (-1)^{S_1} \times (F_1 \pm F_2)\, 2^{E_1}$$

$\Rightarrow$ It has to be normalized



Multiply $\left[(-1)^{S_1} \times F_1 \times 2^{E_1}\right] \times \left[(-1)^{S_2} \times F_2 \times 2^{E_2}\right]$

$$\Rightarrow (-1)^{S_1 \oplus S_2} \times (F_1 \times F_2) \times 2^{E_1 + E_2}$$

$\because 1 \le (F_1 \times F_2) < 4 \Rightarrow$ the result has to be normalized

Divide $\left[(-1)^{S_1} \times F_1 \times 2^{E_1}\right] \div \left[(-1)^{S_2} \times F_2 \times 2^{E_2}\right]$; $F_2 \ne 0$

$$\Rightarrow (-1)^{S_1 \oplus S_2} \times (F_1 \div F_2)\, 2^{E_1 - E_2}$$

$\because 0.5 < (F_1 \div F_2) < 2 \Rightarrow$ the result has to be normalized

Accuracy & Rounding off

- Precision is lost when bits are shifted to the right of rightmost bit
- 3 extra bits are used internally

G (guard) R (round) & S (sticky)

G&R are 2 bits after LSB

S = 1 iff any bit to the right of R is non-zero

$\boxed{1.1\,1101 - - - - - - - 11011GRS}$

Rounding using G, R & S

if G = 1 & R = 1 ; add 1 to LSB

if G = 0 & R = 0 or 1 ; no change

if G = 1 & R = 0 look at 'S'

if S = 1 ⇒ add 1 to LSB

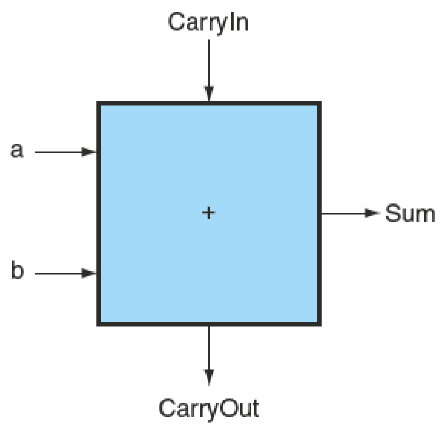
if S = 0 ⇒ round to nearest "even" (i.e., add 1 to LSB, if LSB = 1)

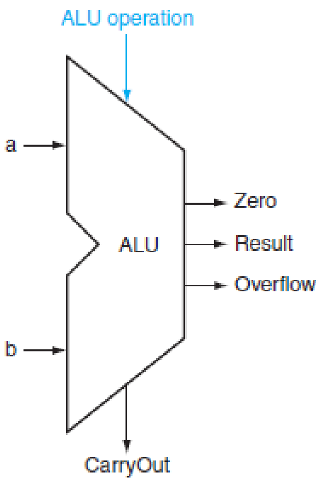
SP E	SP M	DP E	DP M	Object
0	0	0	0	zero
0	NZ	0	NZ	denormal
1-254	any	0-2046	any	normal
255	0	2047	0	infinity
255	nonzero	2047	nonzero	NaN

Arithmetic Circuits

ALU

- 1-bit Adder





Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

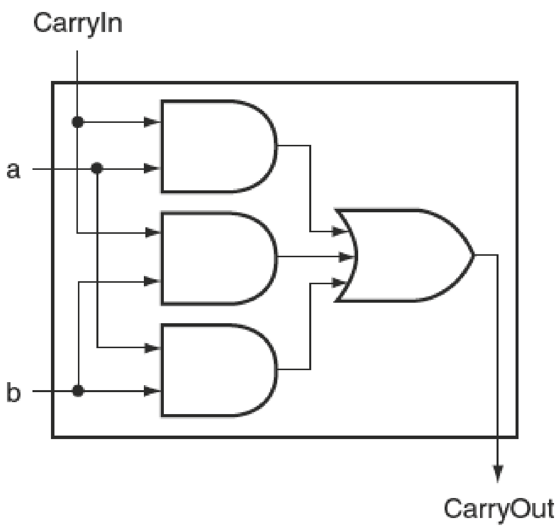
FIGURE B.5.3 Input and output specification for a 1-bit adder.

ALU (Cont..)

$$\text{CarryOut} = (b \cdot \text{CarryIn}) + (a \cdot \text{CarryIn}) + (a \cdot b)$$

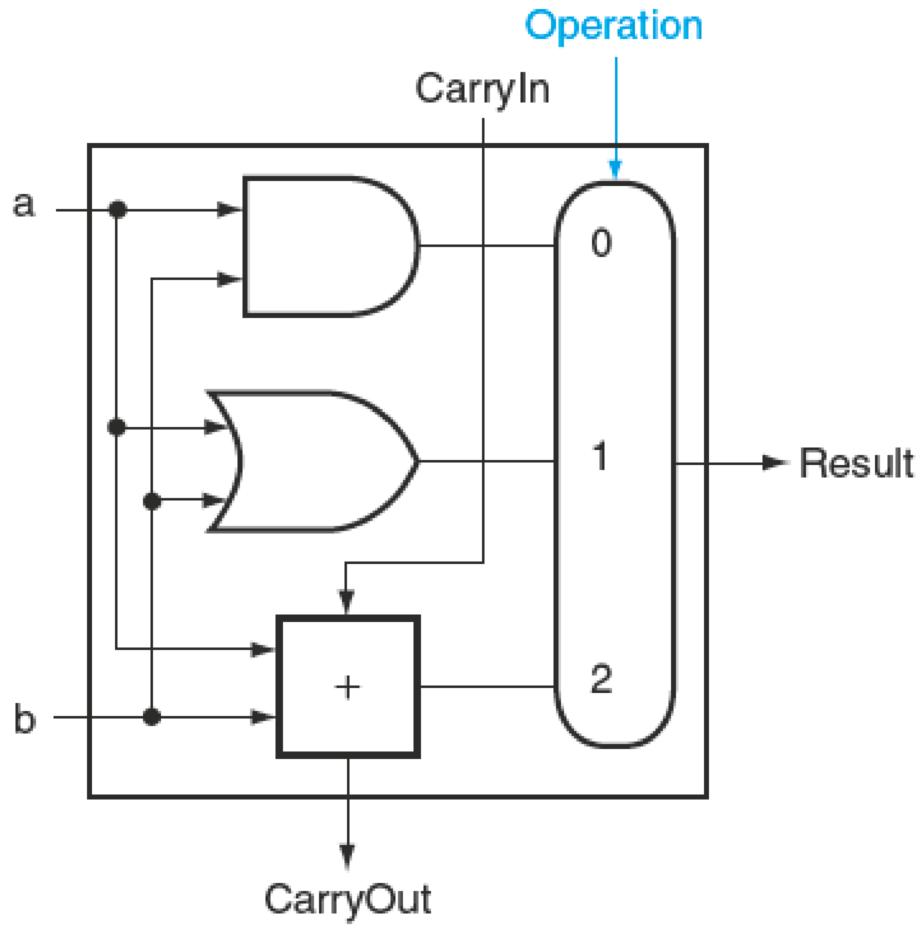
Inputs		
a	**b**	**CarryIn**
0	1	1
1	0	1
1	1	0
1	1	1

FIGURE B.5.4 Values of the Inputs when CarryOut Is a 1.

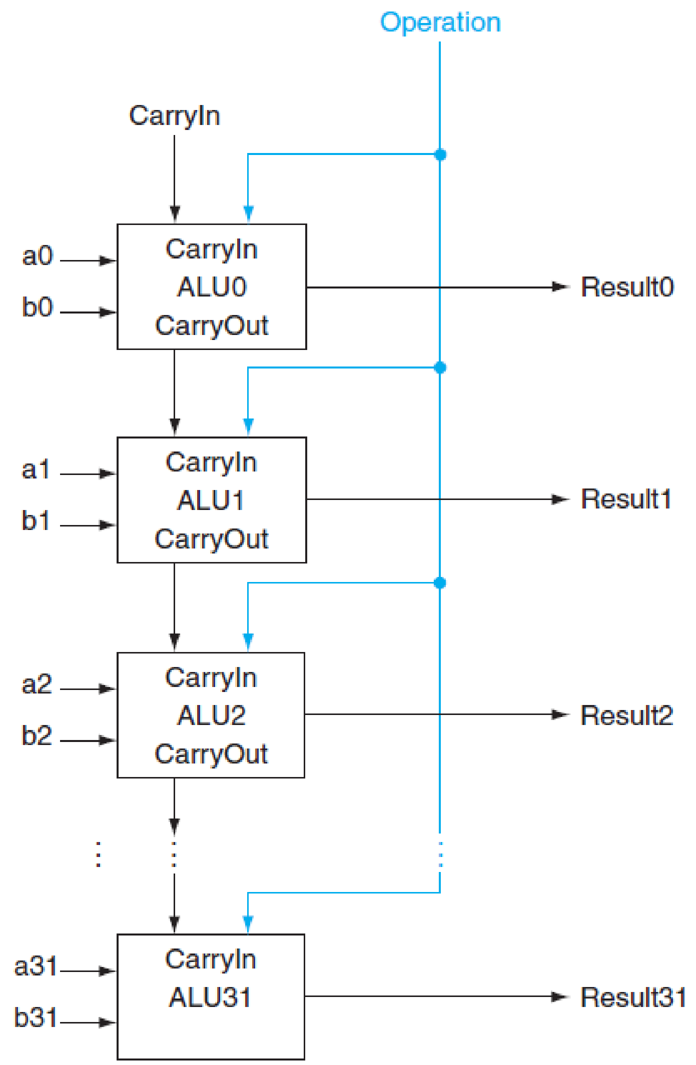


$$\text{Sum} = (a \cdot \overline{b} \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot b \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot \overline{b} \cdot \text{CarryIn}) + (a \cdot b \cdot \text{CarryIn})$$

1-bit ALU (ADD, AND & OR)

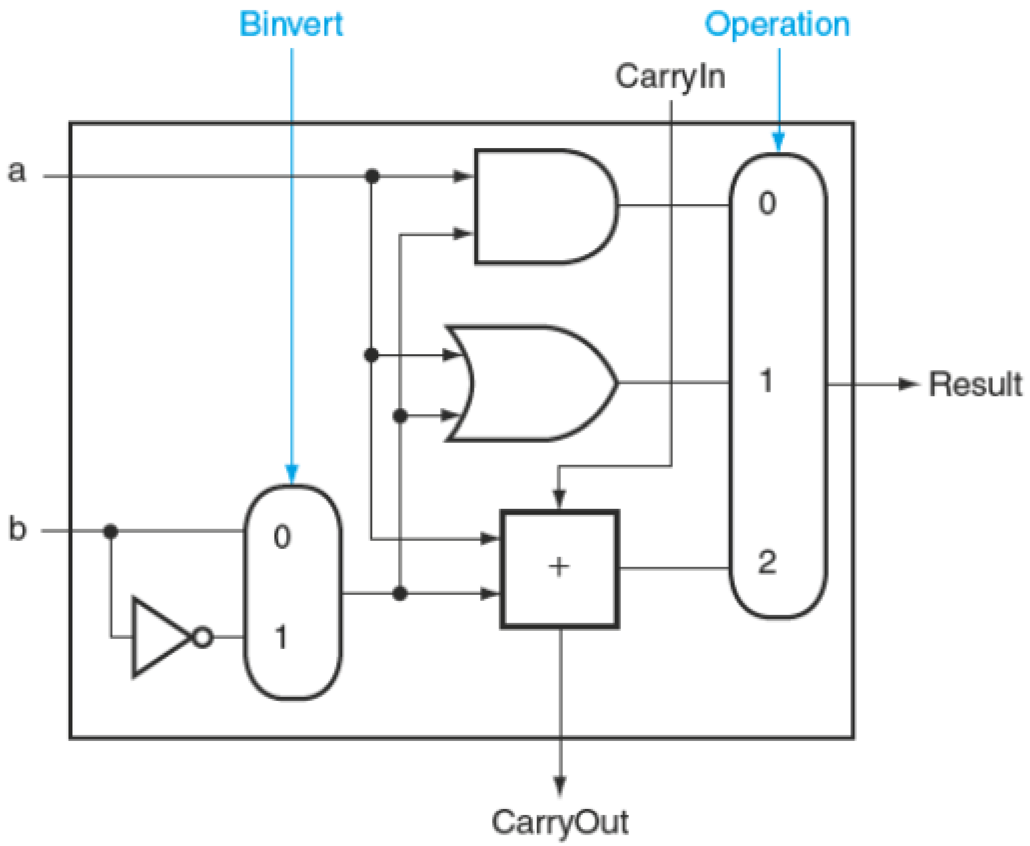


32-bit ALU

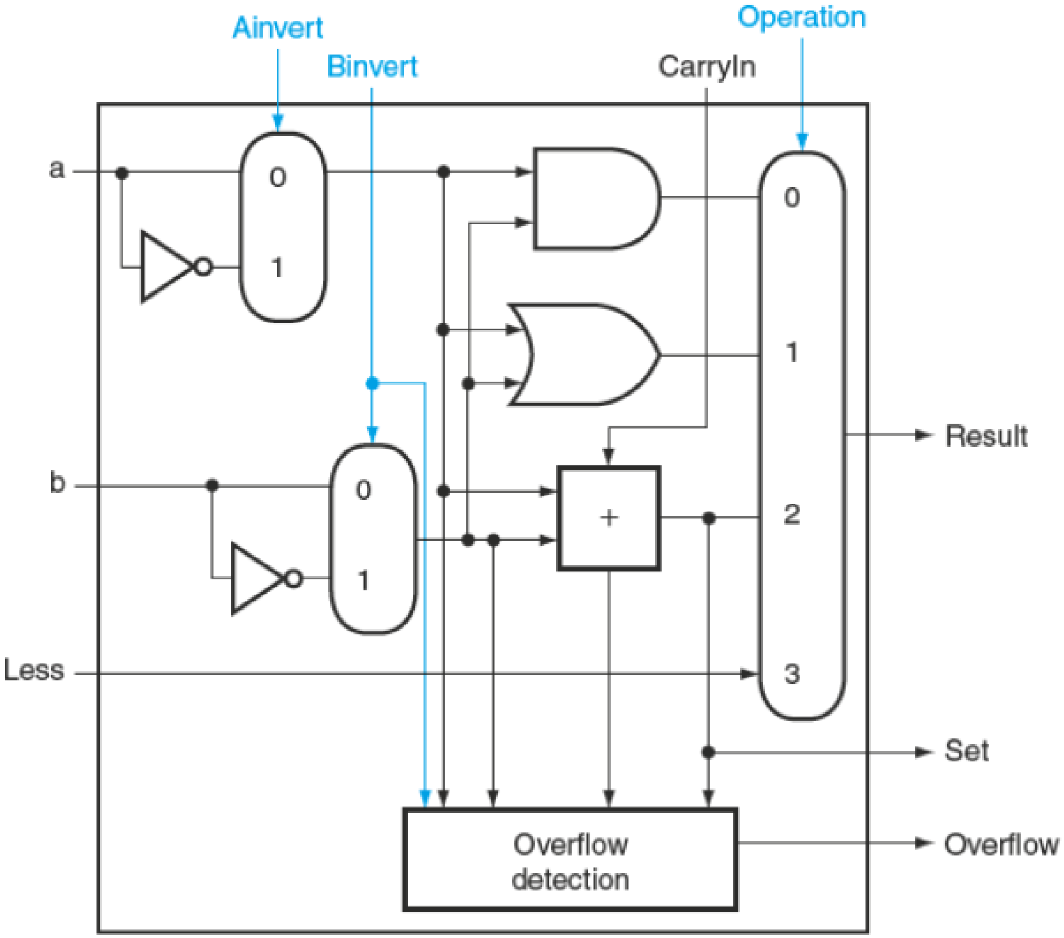


Operation
CarryIn
a0
b0
CarryIn
ALU0
CarryOut
Result0
a1
b1
CarryIn
ALU1
CarryOut
Result1
a2
b2
CarryIn
ALU2
CarryOut
Result2
a31
b31
CarryIn
ALU31
Result31

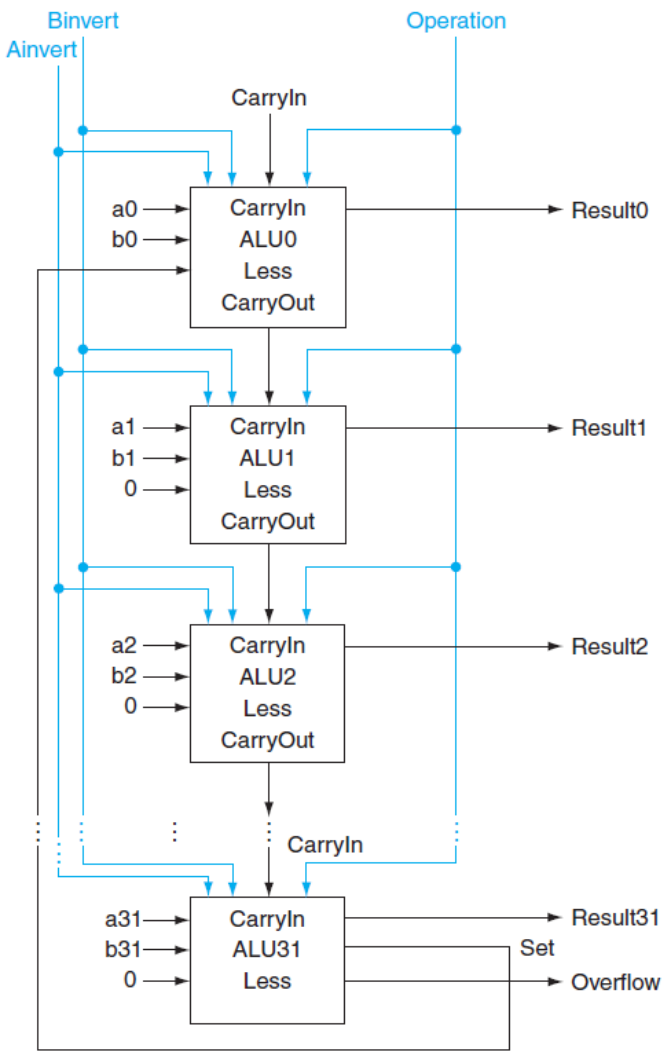
1-bit ALU (ADD, SUB, AND & OR)



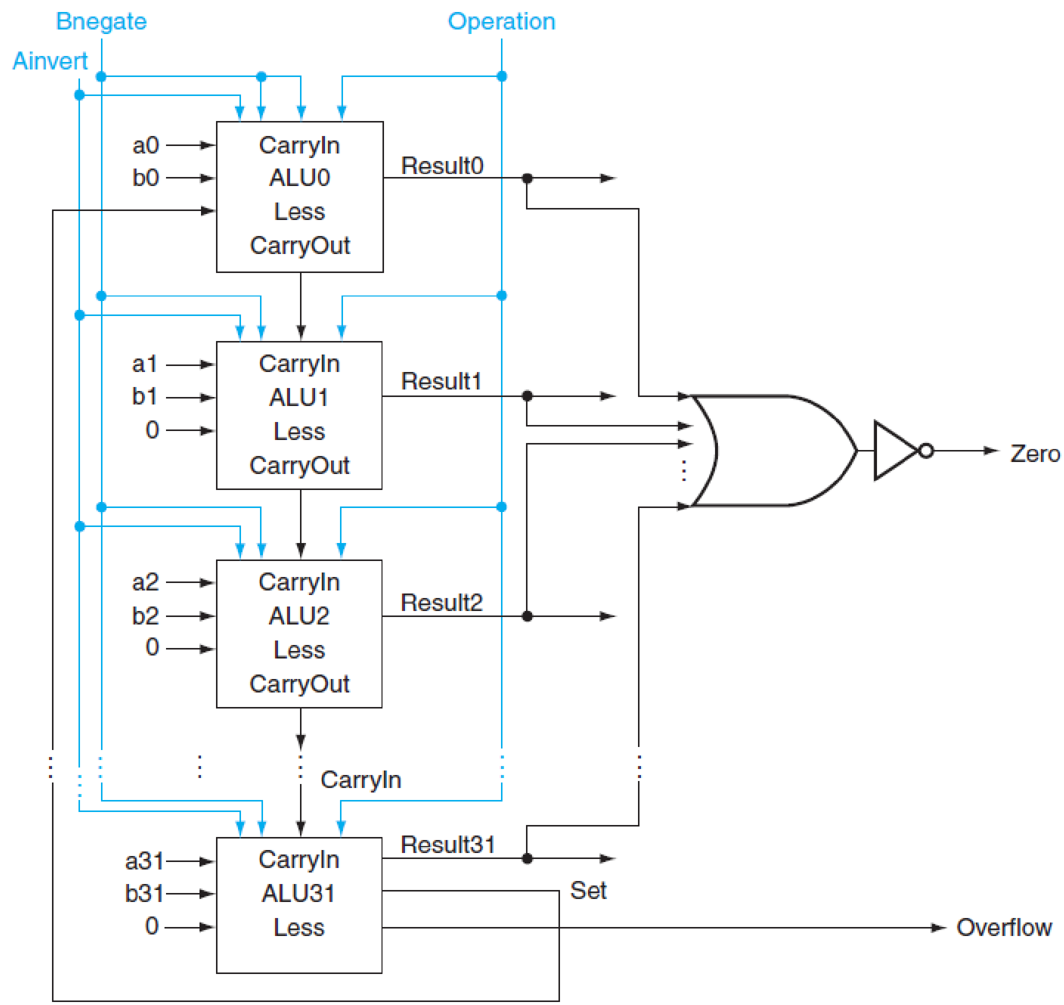
1-bit ALU (ADD, SUB, AND, OR & SLT)



32-bit ALU (ADD, SUB, AND, OR & SLT)



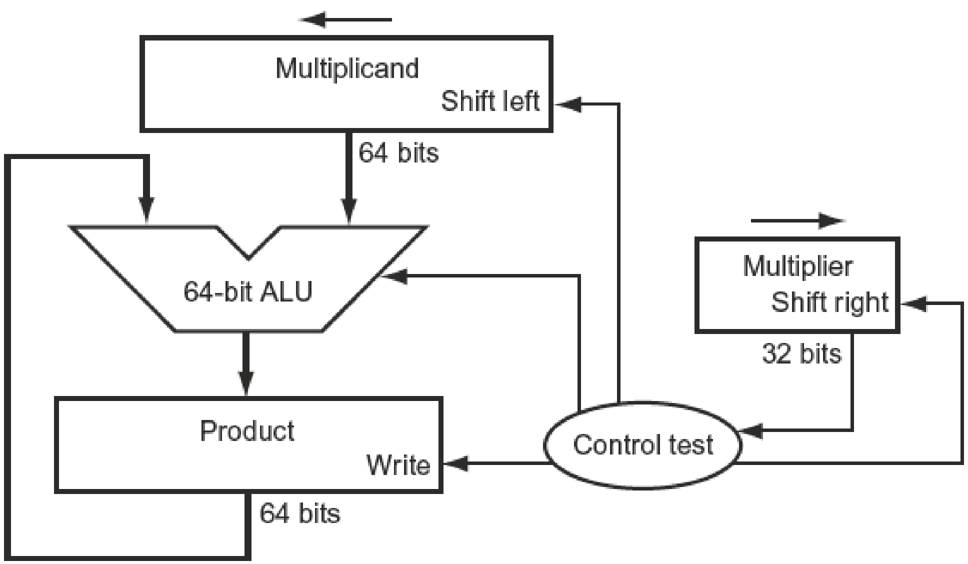
32-bit ALU (ADD, SUB, AND, OR, SLT & EQ)

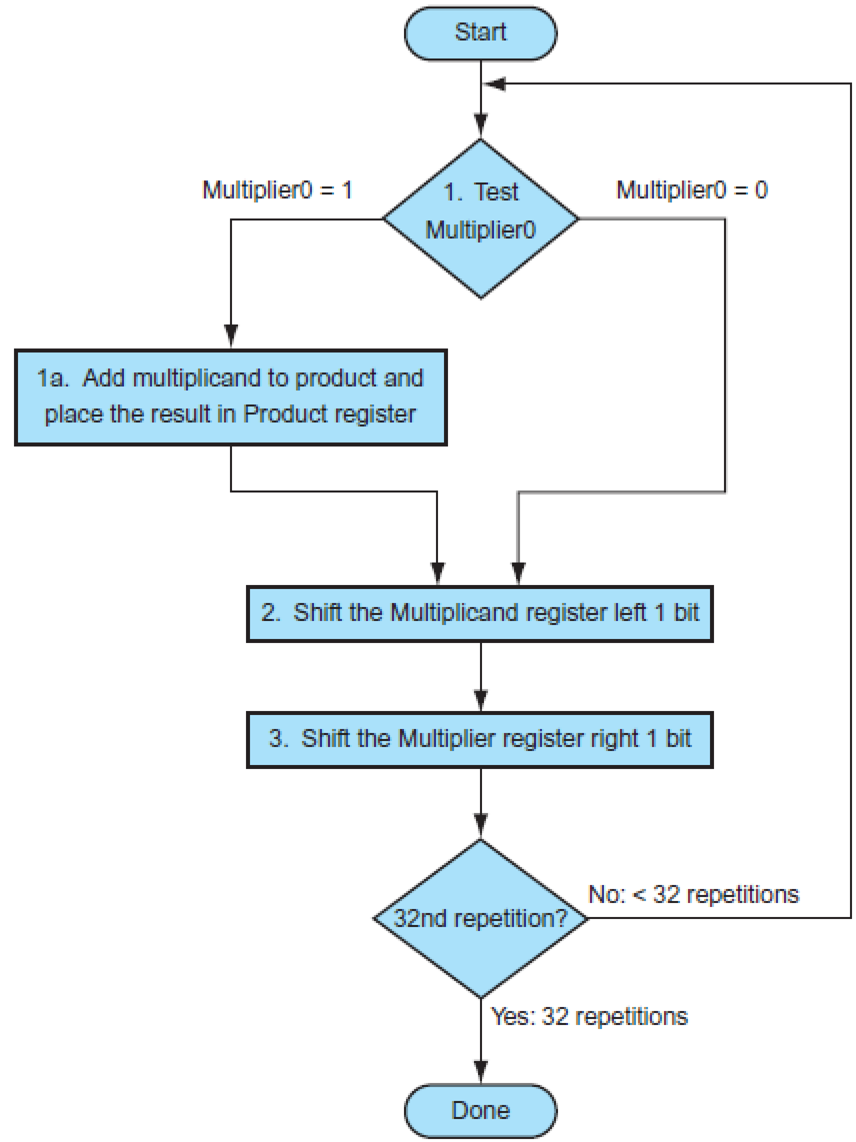


Design of Hardware for Multiplication, Division and Floating point operations

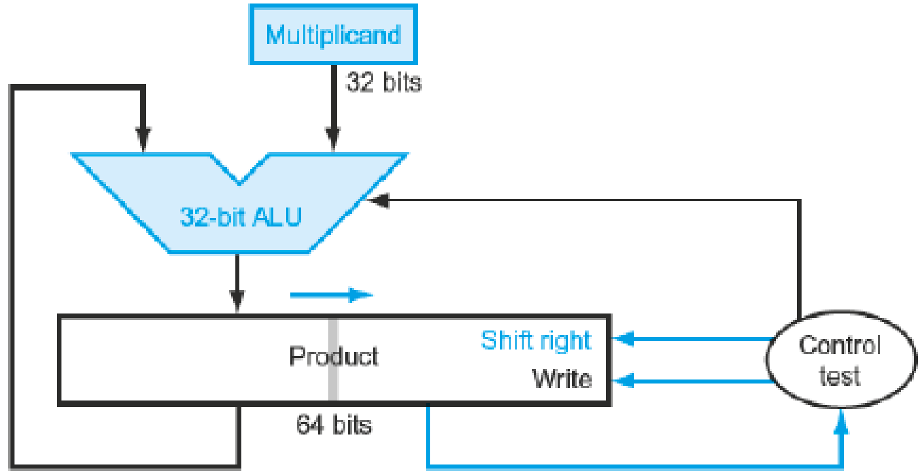
Multiplier

```
Multiplicand        1000ten
Multiplier    x     1001ten
                    -------
                    1000
                   0000
                  0000
                 1000
                 ----------
Product          1001000ten
```



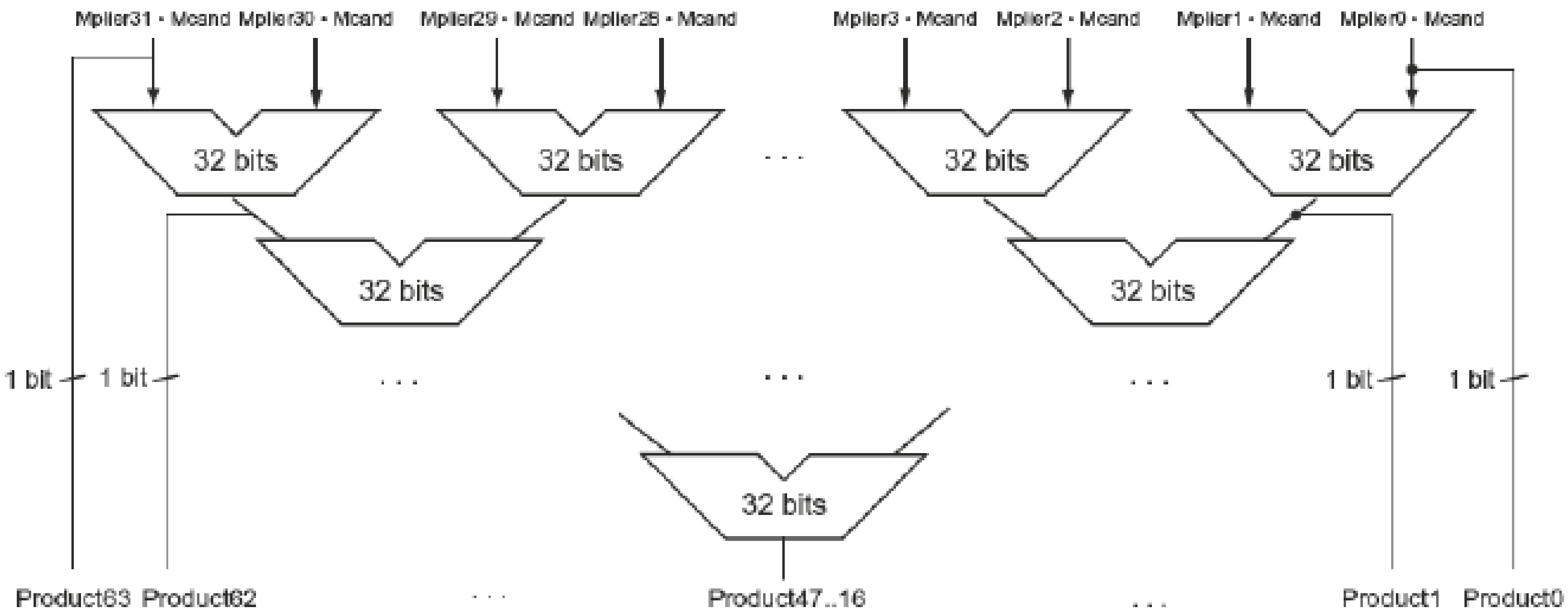


Multiplier (Cont..)

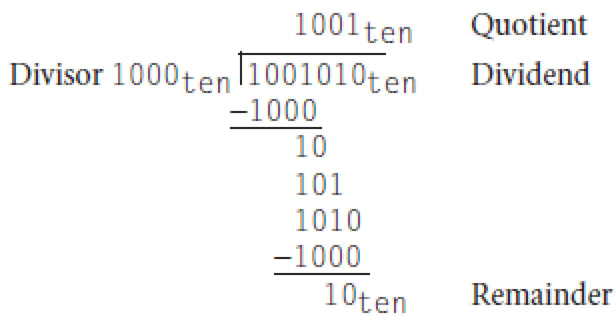


Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⇒ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⇒ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

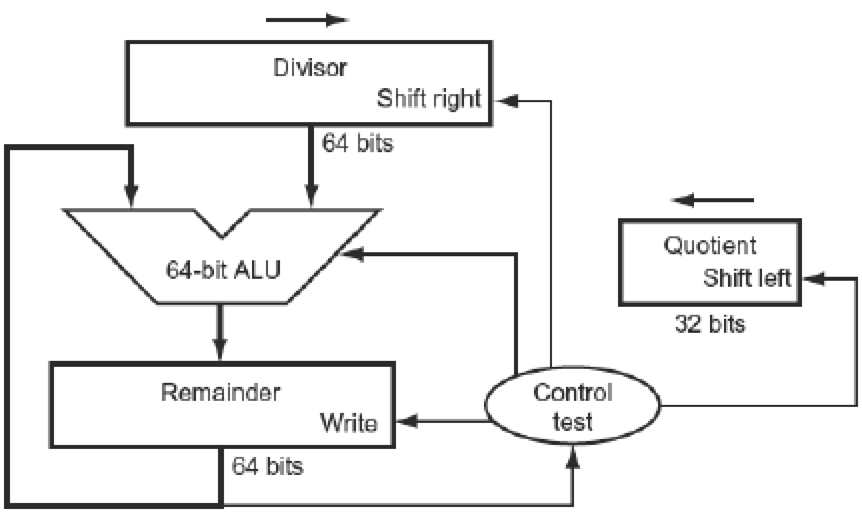
Fast Multiplier

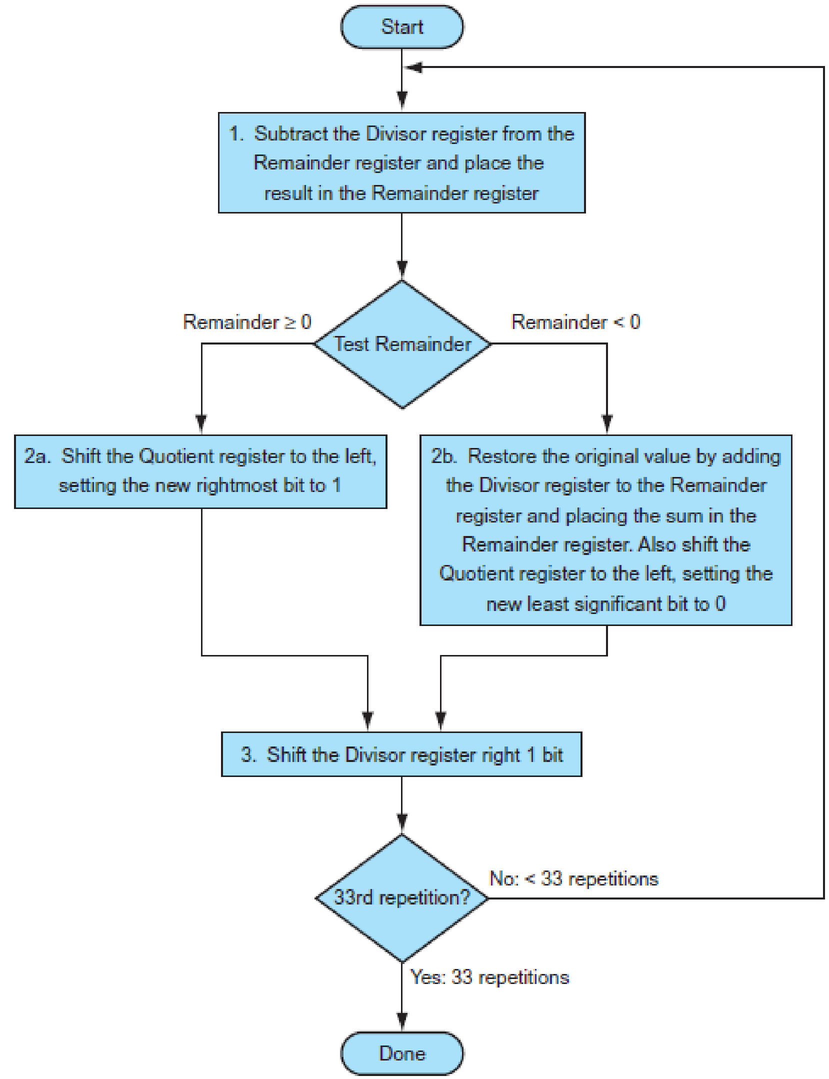


Division

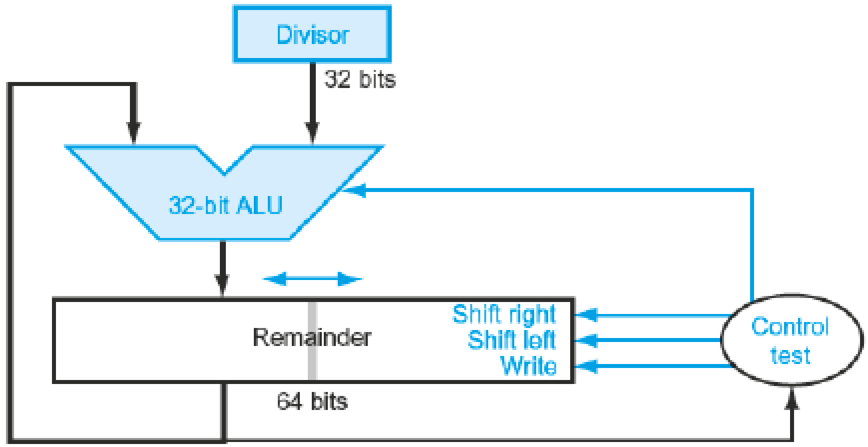


$$\text{Dividend} = \text{Quotient} \times \text{Divisor} + \text{Remainder}$$





Division (Cont..)



Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Floating Point Operations

Single Precession FP Representation (IEEE 754)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent								fraction																						
1 bit	8 bits								23 bits																						

Double Precession FP Representation (IEEE 754)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent												fraction																		
1 bit	11 bits												20 bits																		

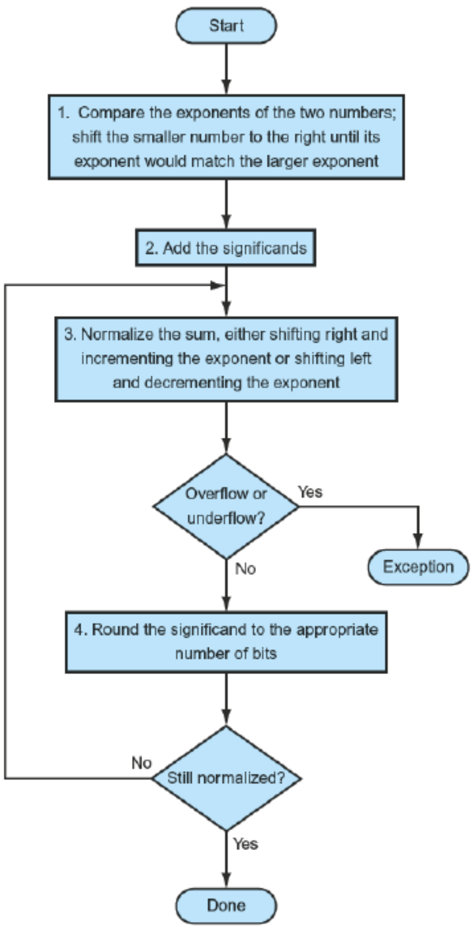
fraction (continued)
32 bits

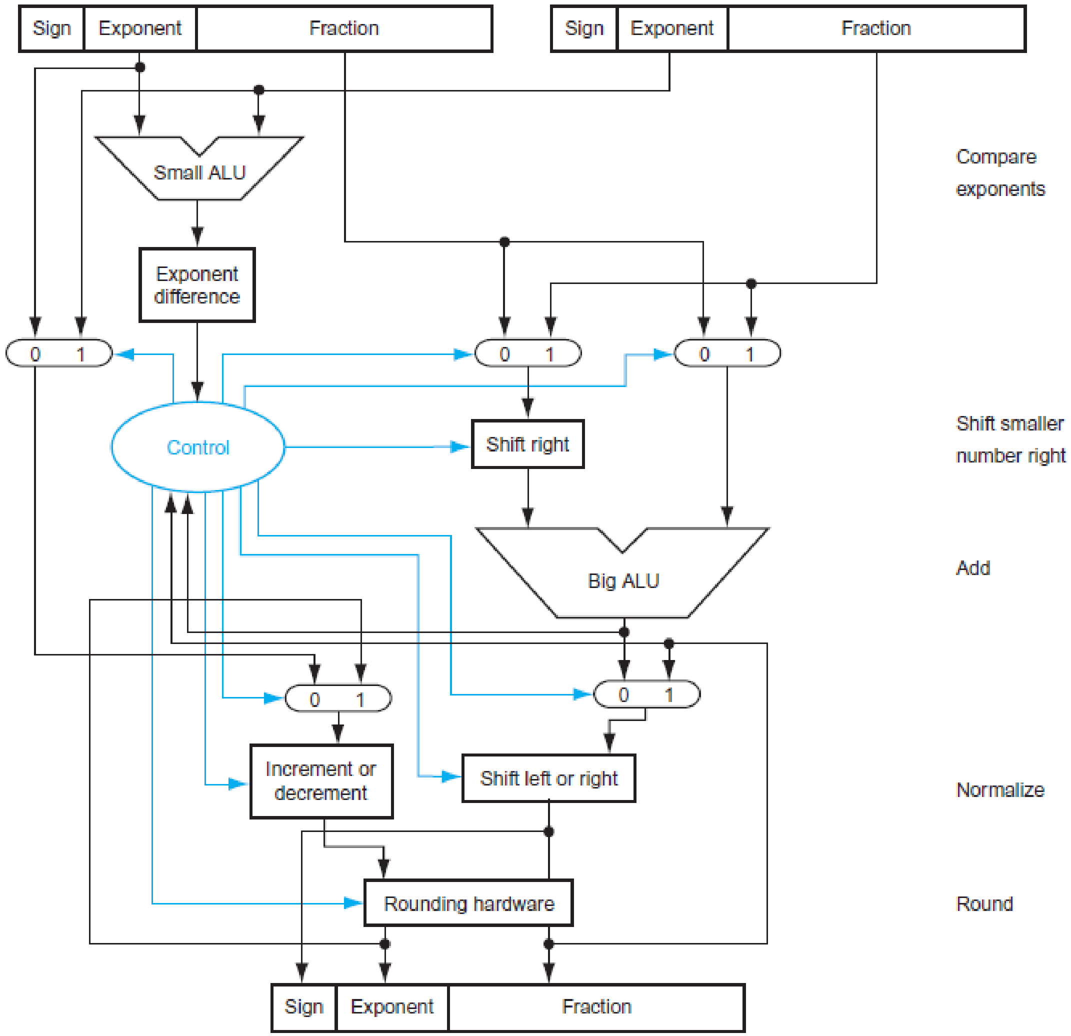
Floating point value

$$(-1)^S \times (1 + \text{Fraction}) \times 2^E$$

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	± denormalized number
1–254	Anything	1–2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

Floating point addition





Floating point Multiplication