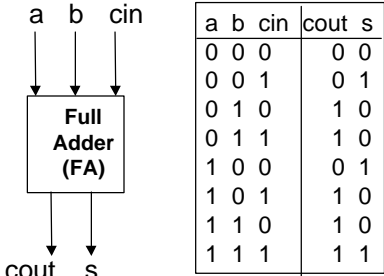


EE 361 ALU

Outline

- Adders
 - Carry look ahead
- Shifters
- Multipliers
 - Unsigned
 - Signed: Booth's algorithm
- Division

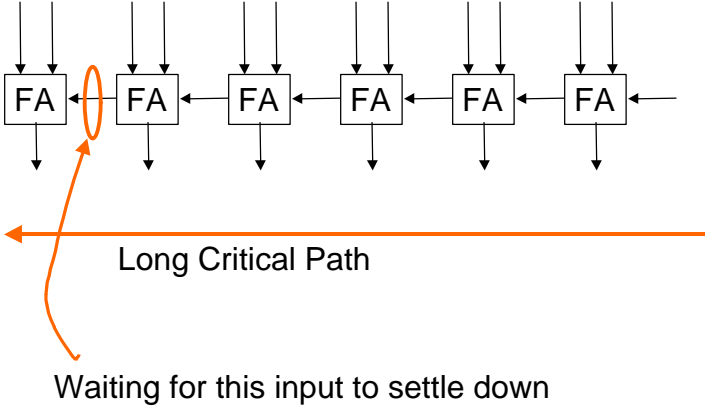
Carry Look Ahead



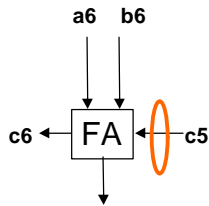
$$\begin{aligned}
 \text{cout} &= ab + bcin + acin \\
 &= ab + (a+b)cin
 \end{aligned}$$

generator propagator

Ripple Adder



Brute Force Approach



$$c5 = a5*b5 + b5*c4 + a5*c4$$

$$= a5*b5 + b5*(a4*b4 + b4*c3 + a4*c3) + \dots$$

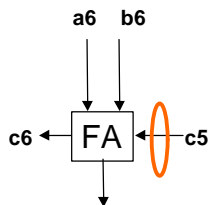
$$= a5*b5 + b5*a4*b4 + b5*b4*c3 + b4*a4*c3 + \dots$$

and so on

= Sum of Product (SOP) expression of product terms that are OR'd together.

Just *two* levels of gates to get c5 BUT its a very large network of gates

Carry Look Ahead



$$c5 = a5*b5 + b5*c4 + a5*c4$$

$$= a5*b5 + (a5+b5)*c4$$

$$= \text{generator} + \text{propagator} * c4$$

$$= g5 + p5*c4$$

$$= g5 + p5*(g4 + p4*c3)$$

$$= g5 + p5*g4 + p5*p4*c3$$

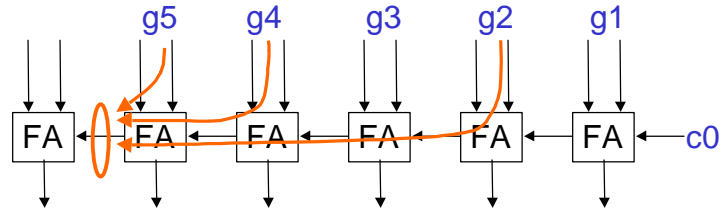
and so on

$$= g5 + p5*g4 + p5*p4*g3 + p5*p4*p3*g2 + \dots$$

$$p5*p4*p3*p2*p1*p0*c0$$

Smaller circuit and 3 levels of gates

Generators and Propagators



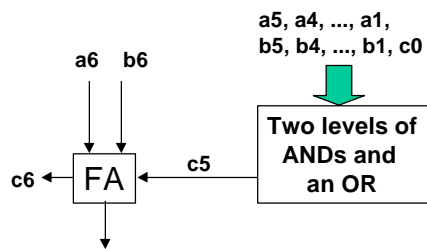
Galen Sasaki

EE 361 University of Hawaii

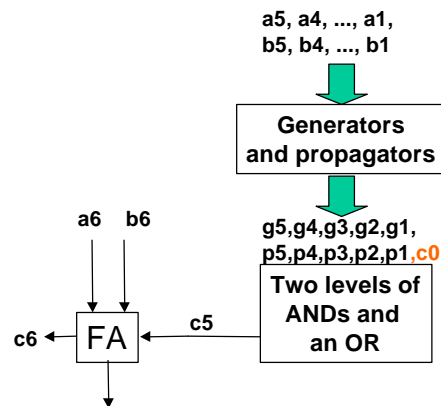
7

Comparison

Brute Force Approach



Carry Look Ahead

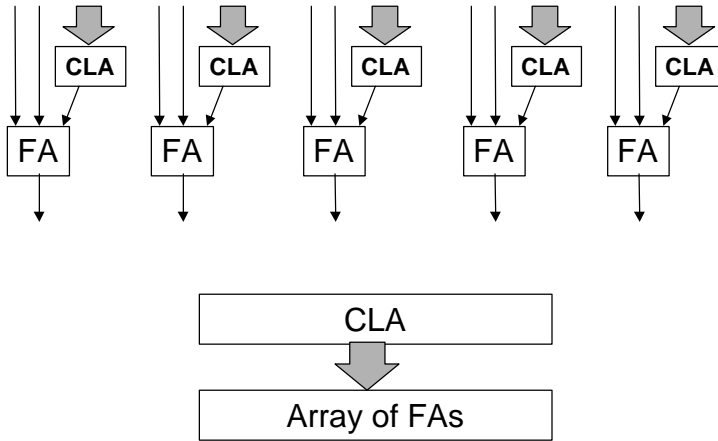


Galen Sasaki

EE 361 University of Hawaii

8

Carry Look Ahead (CLA)

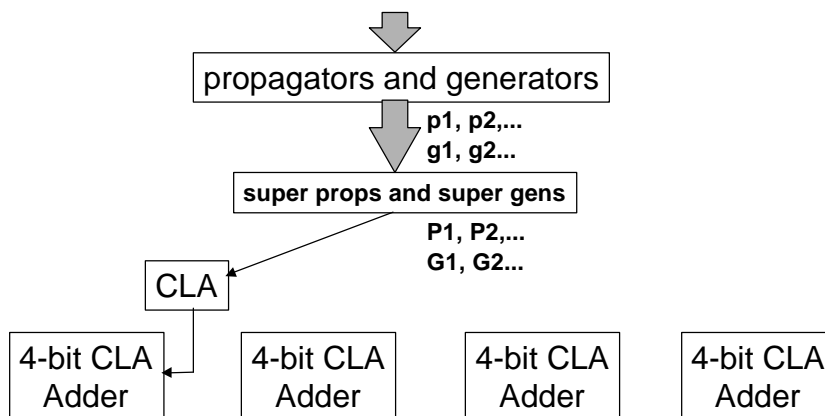


Galen Sasaki

EE 361 University of Hawaii

9

Second Level Of Abstraction



Galen Sasaki

EE 361 University of Hawaii

10

Shifting

- Right or left by one or more bit positions
- Logical shift: shift in zero
 - unsigned multiply/divide by 2 (for single bit shift)
- Rotate: shift in the bit that got shifted out
- Arithmetic shift
 - Left shift = logical left shift
 - Right shift = shift in (copy) sign bit

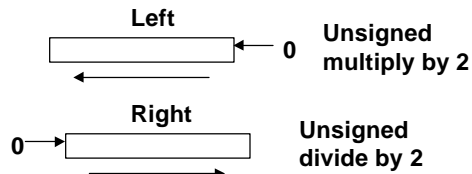
Galen Sasaki

EE 361 University of Hawaii

11

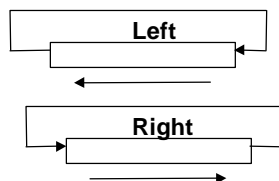
Shifting

Logical shift

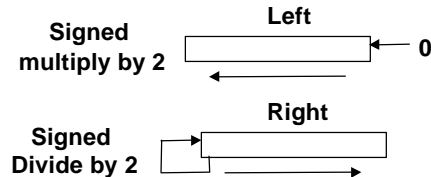


Multiply and divide
Overflow and underflow are possible

Rotate



Arithmetic shift



Galen Sasaki

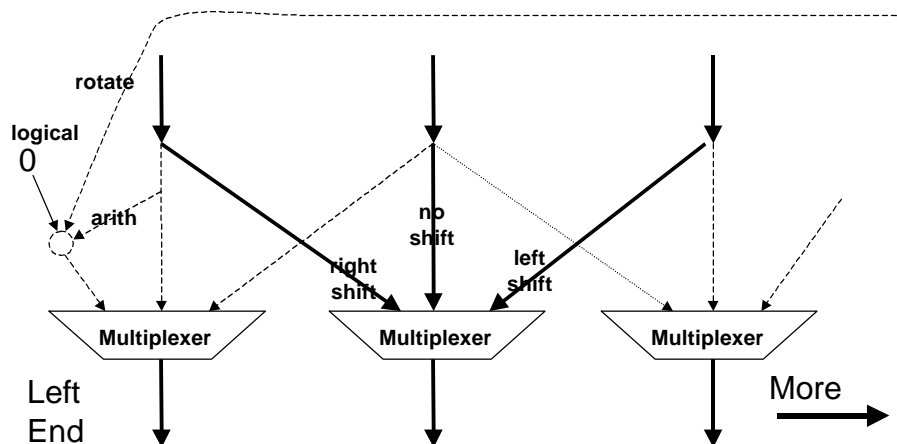
EE 361 University of Hawaii

12

MIPS Instructions

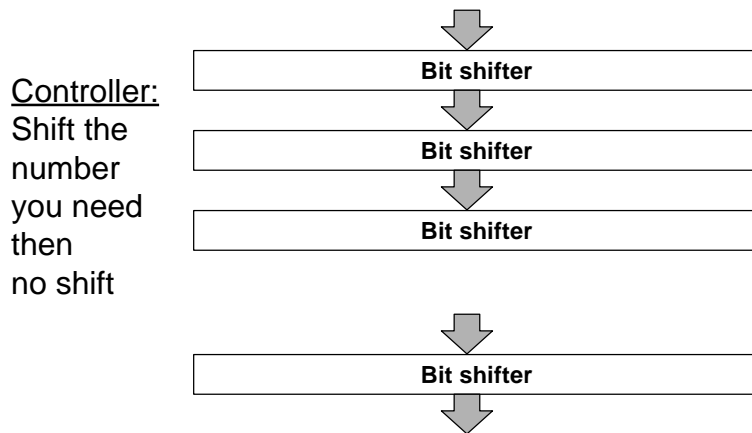
- sll \$r1,\$r2,constant
 - constant = shift amount (shamt)
 - \$r1 = \$r2 << constant
- slr \$r1,\$r2,constant
 - constant = shift amount (shamt)
 - \$r1 = \$r2 << constant

Hardware Shifters



Multiple Shifts

Barrel shifter: it can shift or rotate an input



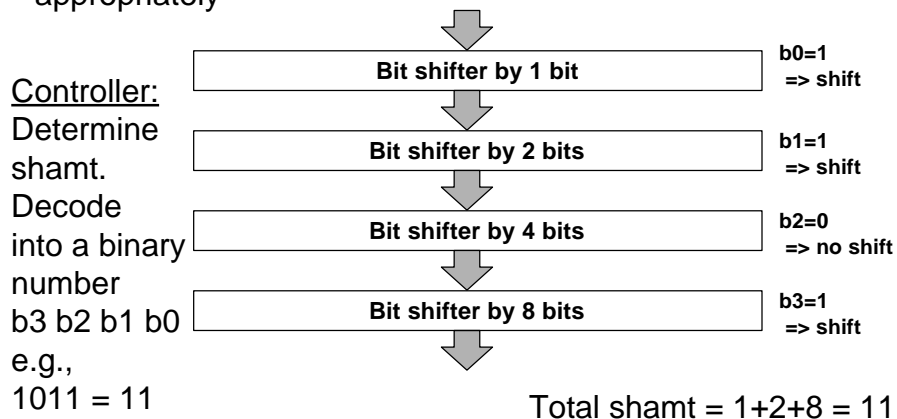
Galen Sasaki

EE 361 University of Hawaii

15

Another Version: less stages

Multiple bit shifts can be done by wiring multiplexers appropriately

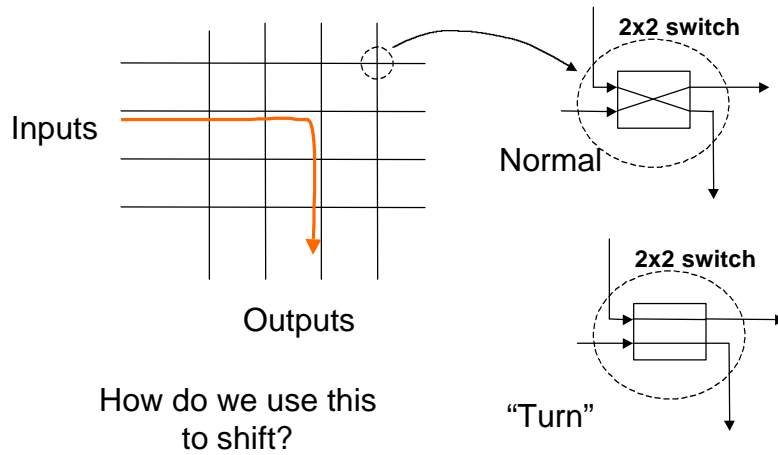


Galen Sasaki

EE 361 University of Hawaii

16

Another Shifter: Crossbar

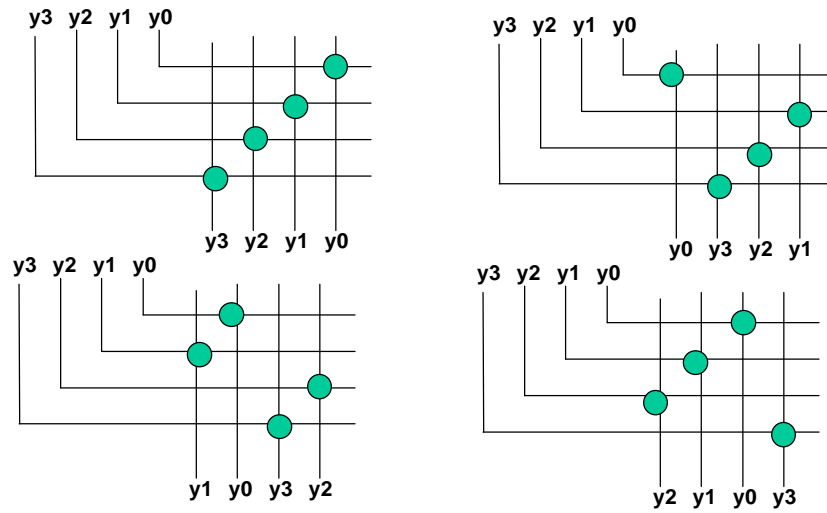


Galen Sasaki

EE 361 University of Hawaii

17

Another Shifter: Crossbar

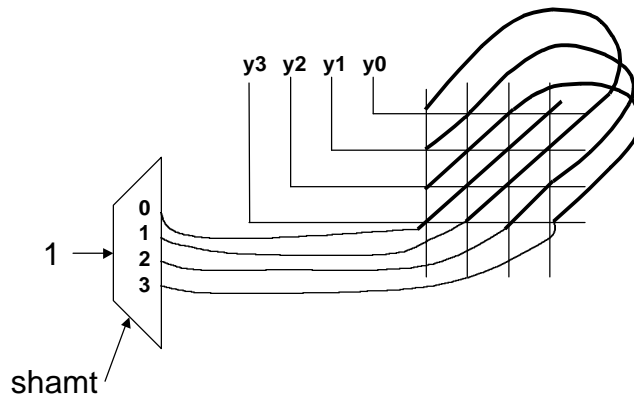


Galen Sasaki

EE 361 University of Hawaii

18

Another Shifter: Controller



Galen Sasaki

EE 361 University of Hawaii

19

Multipliers

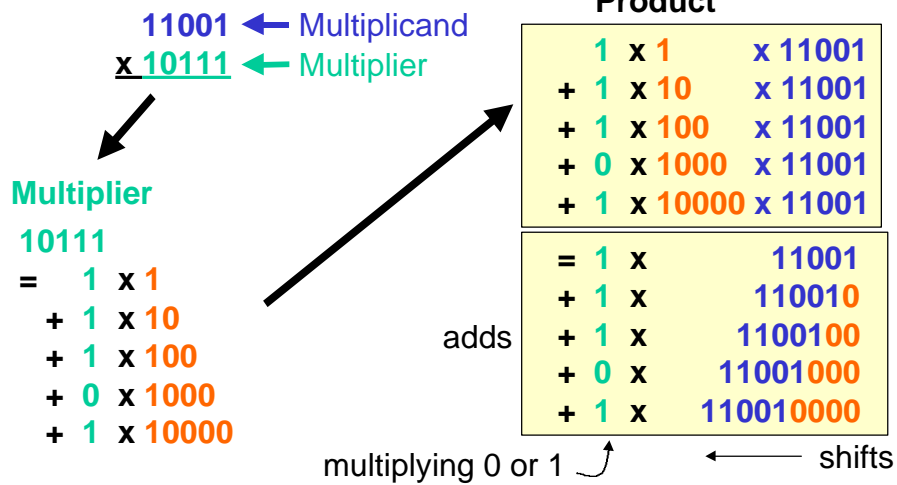
- How do we do it by hand?
- Combinational circuit multiplier
- Unsigned multiplier
 - Versions 1, 2, 3
- What about signed numbers?
- Booth's algorithm
- Hardware

Galen Sasaki

EE 361 University of Hawaii

20

Doing it by hand: decimal #s

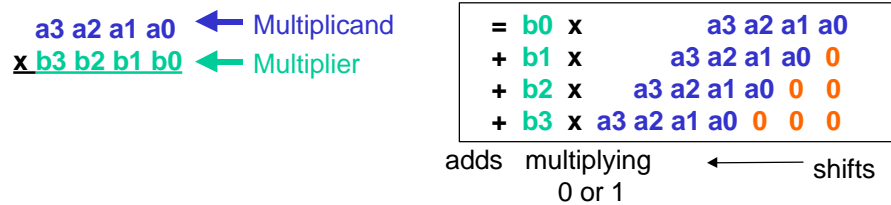


Galen Sasaki

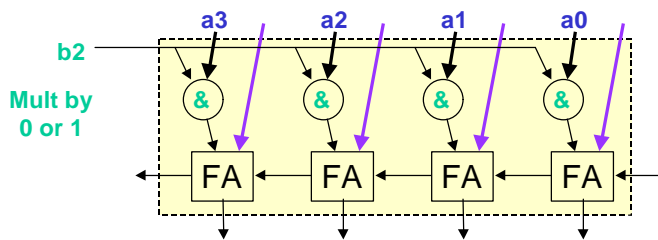
EE 361 University of Hawaii

21

Combinational Circuit



Multiply & Add (M&A)

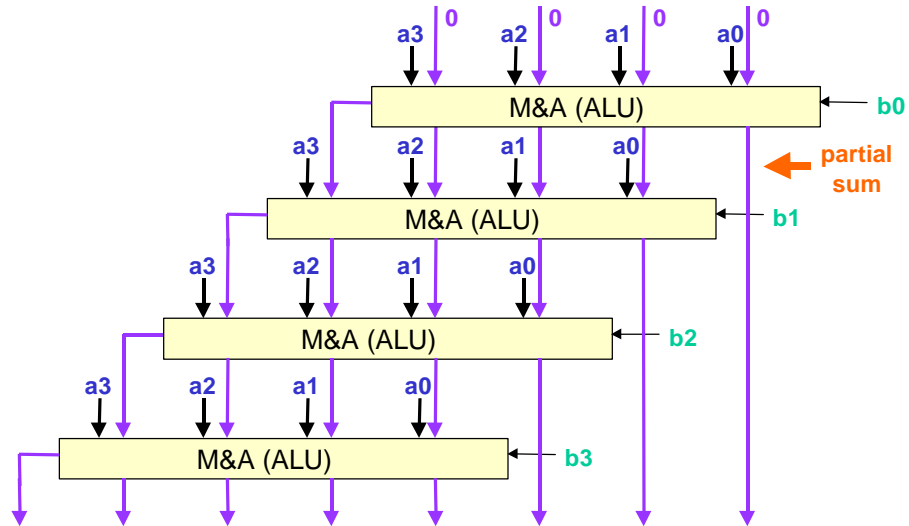


Galen Sasaki

EE 361 University of Hawaii

22

Combinational Circuit



Galen Sasaki

EE 361 University of Hawaii

23

How big is the product

Let's say you multiply

n-bit number x m-bit number

Largest n-bit number = $2^n - 1$

Largest m-bit number = $2^m - 1$

Largest product = $2^{nm} - 2^n - 2^m + 1 < 2^{nm}$

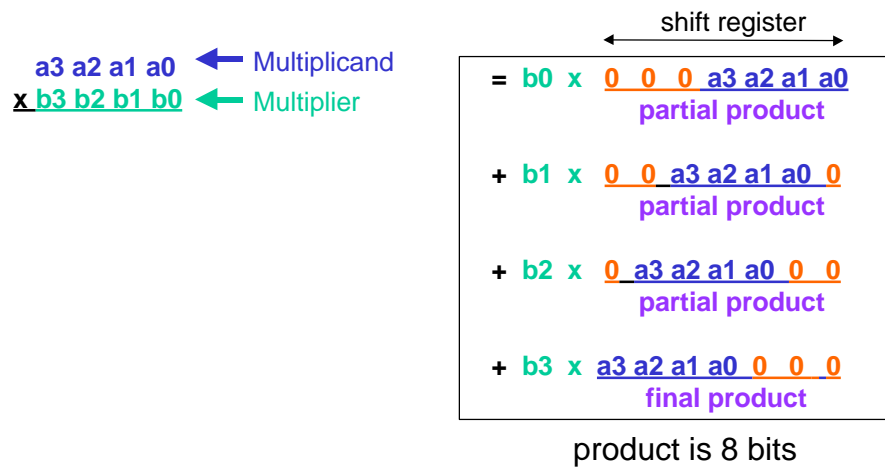
Product can fit in n x m bits

Galen Sasaki

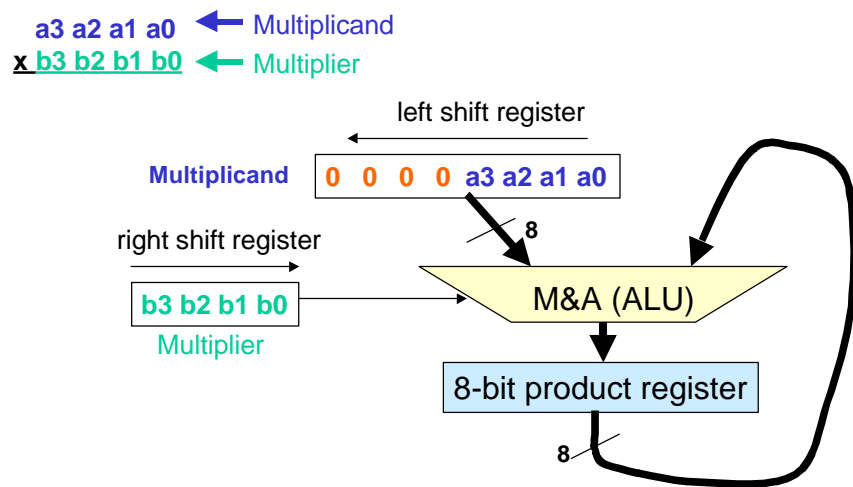
EE 361 University of Hawaii

24

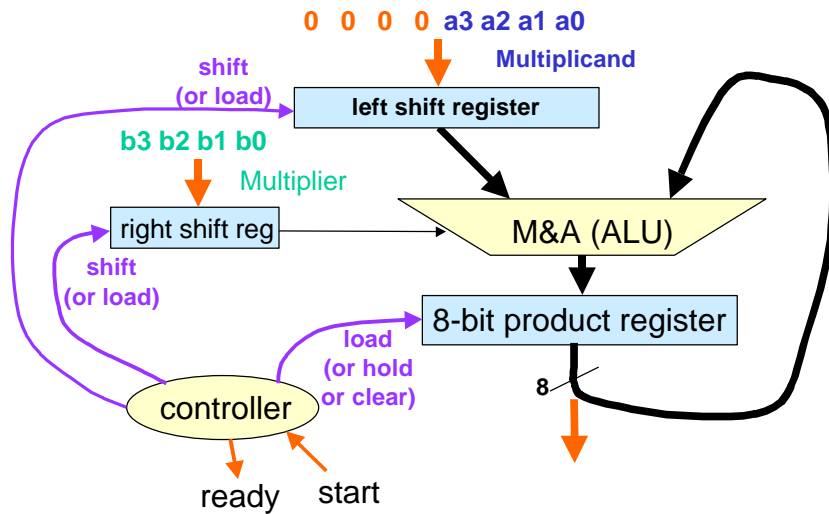
Sequential Circuit



Sequential Circuit



Sequential Circuit

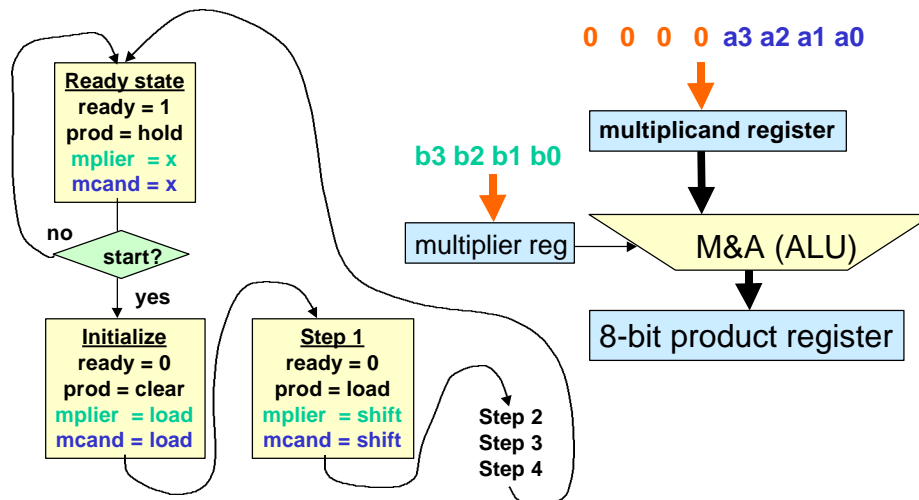


Galen Sasaki

EE 361 University of Hawaii

27

Sequential Circuit: Controller



Galen Sasaki

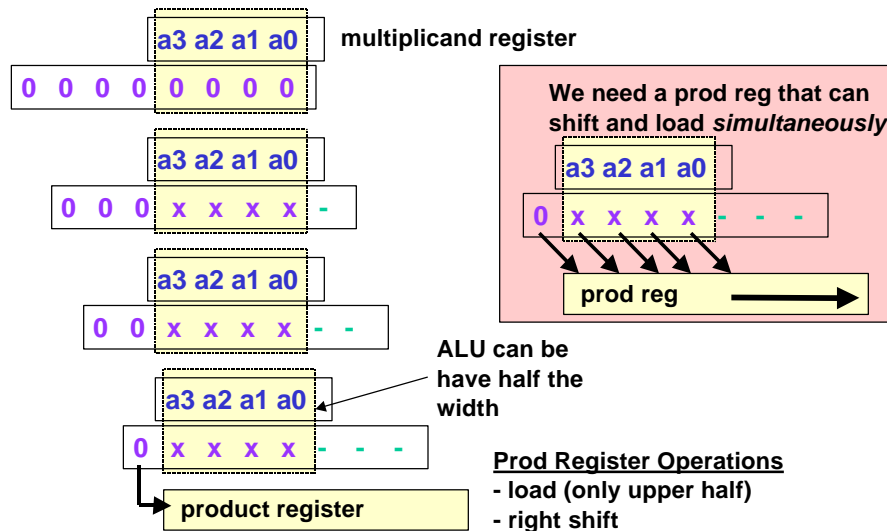
EE 361 University of Hawaii

28

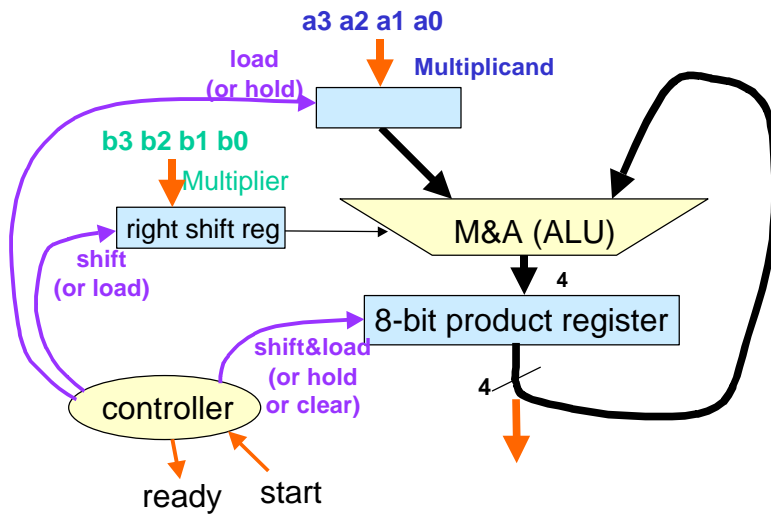
Sequential Circuit: Controller



Improvement



Version 2

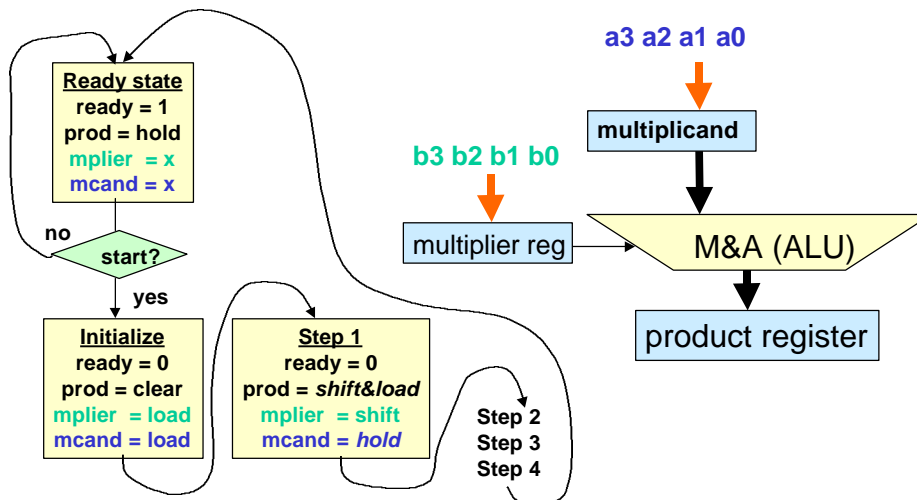


Galen Sasaki

EE 361 University of Hawaii

31

Version 2

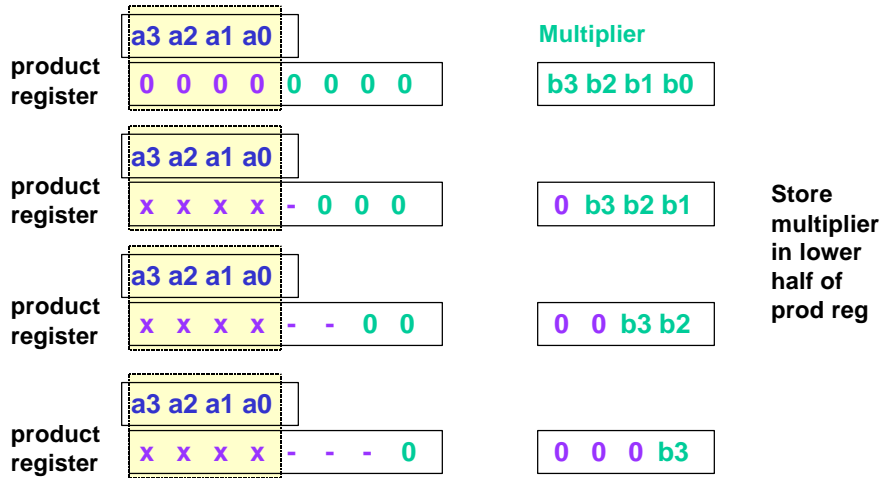


Galen Sasaki

EE 361 University of Hawaii

32

Another Improvement

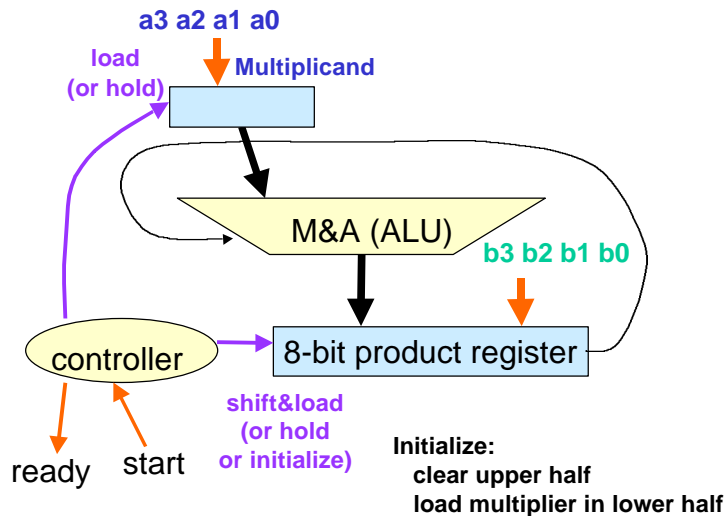


Galen Sasaki

EE 361 University of Hawaii

33

Version 3

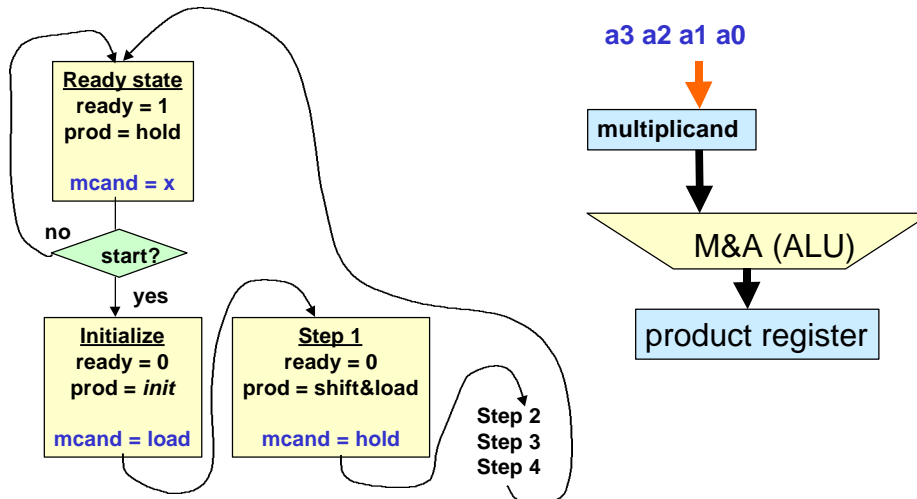


Galen Sasaki

EE 361 University of Hawaii

34

Version 3: Final



Galen Sasaki

EE 361 University of Hawaii

35

Signed Multiplication

- Unsigned multiplication doesn't work for signed numbers
- Booth's algorithm
 - ALU (Multiply & ADD) must be able to multiply by -1 too (subtract)
 - mult by 1: add
 - mult by 0: do nothing
 - mult by -1: sub
 - Looks for strings of 1s in the multiplier

Galen Sasaki

EE 361 University of Hawaii

36

Strings of 0s in multiplier

Example

$$\begin{array}{r}
 a_3 a_2 a_1 a_0 \\
 x 1 0 0 0 1 0 \\
 \hline
 a_3 a_2 a_1 a_0 \\
 a_3 a_2 a_1 a_0
 \end{array}$$

many shifts, few adds

Strings of 1s in multiplier

Example

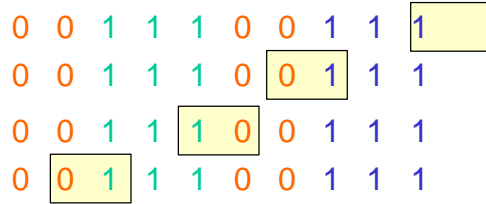
$$\begin{array}{r}
 a_3 a_2 a_1 a_0 \\
 x 0 1 1 1 1 0 \\
 \hline
 = \\
 a_3 a_2 a_1 a_0 \\
 x \\
 [+ 1 0 0 0 0 0 \\
 - 0 0 0 0 1 0
 \end{array}$$

OR

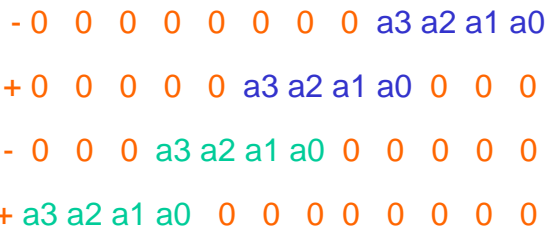
$$\begin{array}{r}
 a_3 a_2 a_1 a_0 \\
 - a_3 a_2 a_1 a_0 \\
 a_3 a_2 a_1 a_0 \\
 + a_3 a_2 a_1 a_0 \\
 a_3 a_2 a_1 a_0
 \end{array}$$

Right after a run Start of a run

Example

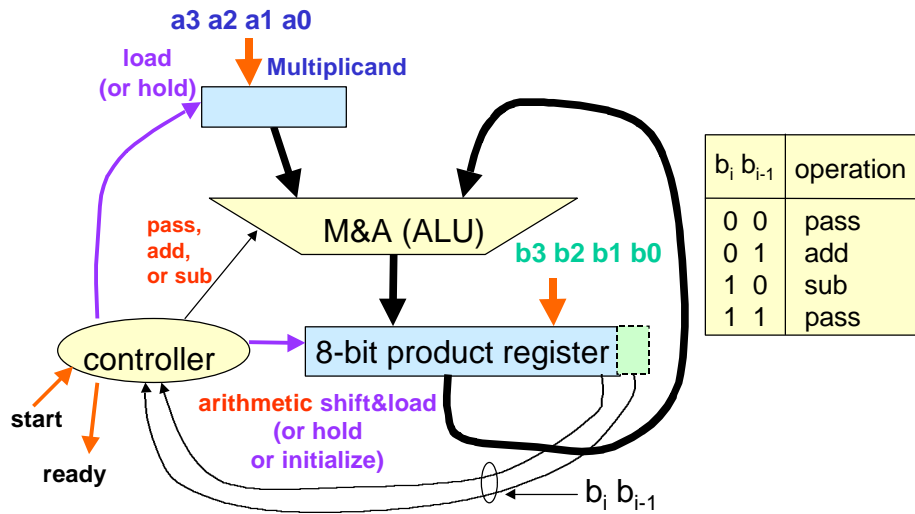


b_i	b_{i-1}	operation
0	0	nothing
0	1	add
1	0	sub
1	1	nothing



b_{i-1}	b_i	operation
-1		sub
+1		add
0		nothing

Hardware: modified ver. 3



b_i	b_{i-1}	operation
0	0	pass
0	1	add
1	0	sub
1	1	pass

Why it works for signed #s

a × b Binary rep of b: $b_{n-1}b_{n-2}...b_0$ Value: $-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$

Booth's Algorithm

$$b_{i-1} - b_i = \begin{cases} 0 & \text{pass} \\ +1 & \text{add} \\ -1 & \text{sub} \end{cases}$$

Product

$$\begin{aligned} &(b_{-1} - b_0) \times a \times 2^0 \\ &+ (b_0 - b_1) \times a \times 2^1 \\ &+ (b_1 - b_2) \times a \times 2^2 \\ &\dots \\ &+ (b_{n-2} - b_{n-1}) \times a \times 2^{n-1} \end{aligned}$$

$$\begin{aligned} &= b_{-1} \times a \\ &+ (2^1 - 2^0) \times b_0 \times a \\ &+ (2^2 - 2^1) \times b_1 \times a \\ &\dots \\ &+ (2^{n-1} - 2^{n-2}) \times b_{n-2} \times a \\ &- 2^{n-1} \times b_{n-1} \times a \end{aligned}$$

Why it works for signed #s

a × b Value of b: $-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$

Product

$$\begin{aligned} &= +(2^1 - 2^0) \times b_0 \times a \\ &+ (2^2 - 2^1) \times b_1 \times a \\ &\dots \\ &+ (2^{n-1} - 2^{n-2}) \times b_{n-2} \times a \\ &- 2^{n-1} \times b_{n-1} \times a \end{aligned}$$

$$\begin{aligned} &= \left(-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \right) \times a \\ &= b \times a \end{aligned}$$

Why it works for signed #s

a x b Value of b: $-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$

Why? Check two cases

Case 1: $b_{n-1} = 0$

$$0 + \sum_{i=0}^{n-2} b_i 2^i$$

↑
Value of b

Case 2: $b_{n-1} = 1$

negative #

$b_{n-1}b_{n-2}\dots b_0$

is the unsigned
binary rep of

$$2^n - |b|$$

value of b

$$\sum_{i=0}^{n-1} b_{i-1} 2^{i-1} = 2^n - |b|$$

$$-|b| = -2^n + \sum_{i=0}^{n-1} b_{i-1} 2^{i-1}$$

$$= -2^n + 2^{n-1} + \sum_{i=0}^{n-2} b_{i-1} 2^{i-1}$$

$$= -2^{n-1} + \sum_{i=0}^{n-2} b_{i-1} 2^{i-1}$$

Division

- How do we do it by hand?
- Hardware circuit
 - Version 1
 - Version 2
 - Version 3
- Signed division

Division

Dividend
Divisor

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

$$\text{Remainder} < \text{Divisor}$$

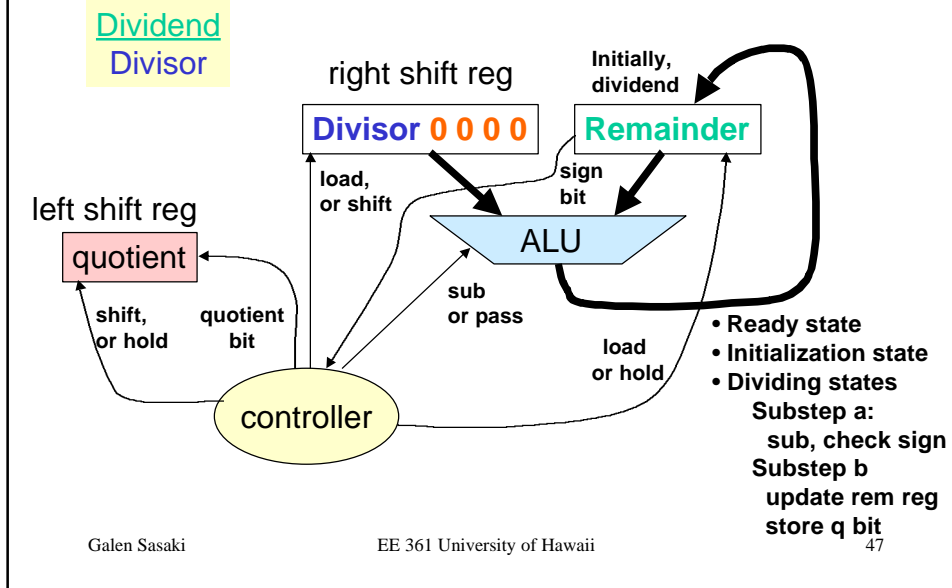
$ \begin{array}{r} 1001 \overline{) 110001} \\ \underline{-1001} \\ 1101 \\ \underline{-1001} \\ 1101 \\ \underline{-1001} \\ 100 \end{array} $	<p style="text-align: right;"><u>quotient</u></p> <p>smaller? yes then subtract: q = 1</p> <p>smaller? no then don't sub: q = 0</p> <p>smaller? yes then subtract: q = 1</p>
<p>remainder</p>	<p>least sig bit</p>

Division

Dividend
Divisor

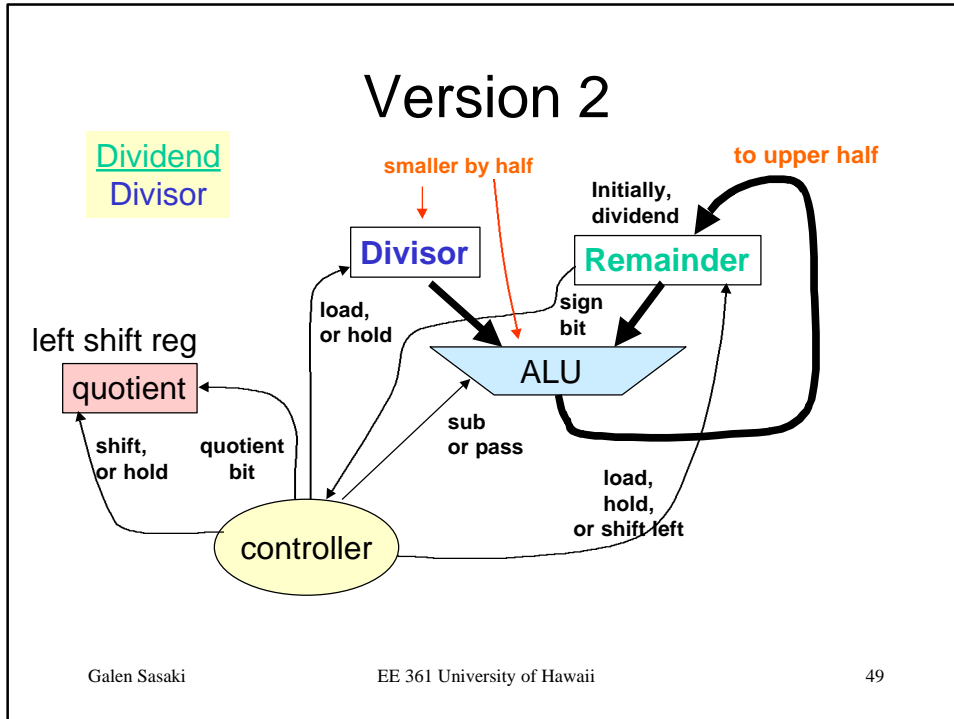
$ \begin{array}{r} \text{Dividend} \\ - \text{Divisor } 0000 \\ \hline \text{remaining dividend} \\ - \text{Divisor } 000 \\ \hline - \text{Divisor } 00 \\ \hline - \text{Divisor } 0 \\ \hline - \text{Divisor} \\ \hline \text{remainder} \end{array} $	<p style="text-align: center;">right shift register</p> <p>smaller? yes then subtract: q = 1 no then don't sub: q = 0</p> <div style="border: 1px solid black; display: inline-block; padding: 2px;">left shift reg</div>
--	---

Hardware



Controller

- Step 1. $\text{Rem reg} = \text{rem reg} - \text{divisor reg}$
- Step 2.
 - 2a. if $\text{rem} \geq 0$ then $\text{quo reg} \leftarrow 1$
 - 2b. else ($\text{rem} < 0$)
 - $\text{rem reg} = \text{rem reg} + \text{divisor reg}$ (restore value)
 - $\text{quo reg} \leftarrow 0$
- Step 3. $\text{Divisor reg} \gg 1$
- Step 4. If repetition = 5 then STOP, else go to Step 1

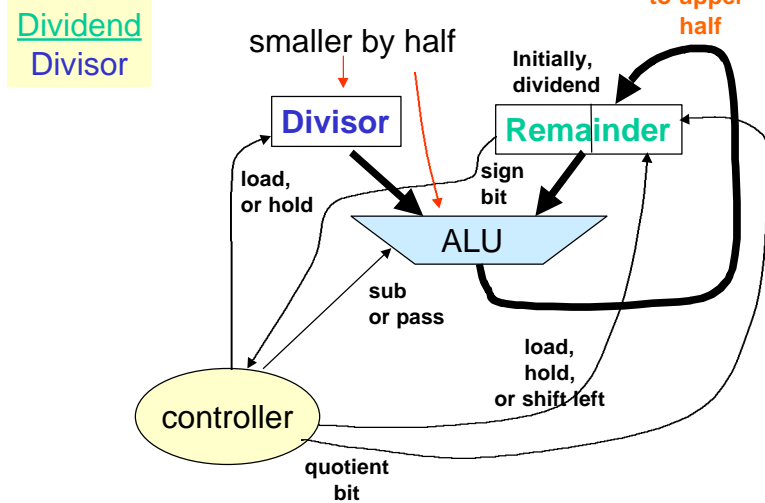


- ## Controller
- Step 1. $\text{Rem reg} = \text{rem reg} - \text{divisor reg}$
 - Step 2.
 - 2a. if $\text{rem} \geq 0$ then $\text{quo reg} \leftarrow 1$
 - 2b. else ($\text{rem} < 0$)
 - $\text{rem reg} = \text{rem reg} + \text{divisor reg}$ (restore value)
 - $\text{quo reg} \leftarrow 0$
 - Step 3. **Rem reg $\ll 1$ (except last pass)**
 - Step 4. If repetition = **5** then STOP, else go to Step 1
 - One too many quotient bits
 - First pass won't produce a sub
- Galen Sasaki EE 361 University of Hawaii 50

Controller

- Step 1. **Rem reg** $\ll 1$
- Step 2. **Rem reg** = rem reg - divisor reg
- Step 3.
 - 3a. if rem ≥ 0 then quo reg $\leftarrow 1$
 - 3b. else (rem < 0)
 - rem reg = rem reg + divisor reg (restore value)
 - quo reg $\leftarrow 0$
- Step 4. If repetition = **4** then STOP, else go to Step 1

Version 3



Controller

- Step 1. Rem reg $\ll 1$
- Step 2. Rem reg = rem reg - divisor reg
- Step 3.
 - 3a. if rem ≥ 0 then
 - ALU: pass through
 - Rem register: shift in 1 (for quotient)
 - 3b. else (rem < 0)
 - ALU: rem reg = rem reg + divisor reg (restore value)
 - Rem register: shift in 0 (for quotient)
- Step 4. If repetition = 4 then STOP, else go to Step 2
- Step 5. Shift left half of rem register by 1 bit

Signed Division

Dividend/Divider

- Remember signs of dividend and divider
- Sign of quotient = "-" if sign dividend \neq sign divider
- Sign of remainder?
 - Sign of remainder = sign of dividend

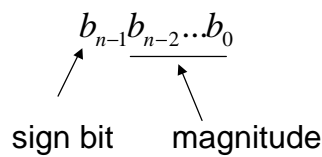
Dividend = Quotient x Divider + Remainder

-7 / +2 Quotient = -3 Remainder = -1
OR
Quotient = -4 Remainder = +1 ???

Floating Point

- Sign-Magnitude arithmetic
- Fixed point non-integer representation
- Scientific notation
 - Arithmetic
- Floating point
- IEEE floating point representation
 - Arithmetic
 - Minimizing errors

Sign Magnitude



Comparison

+7

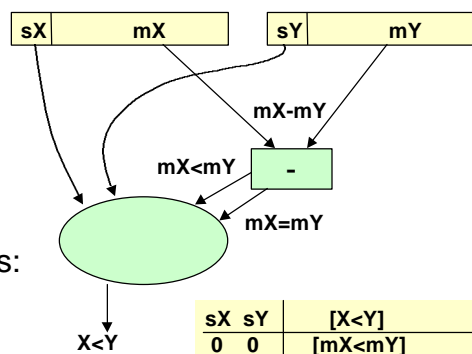
-5

check signs:
" + " > " - "

-4

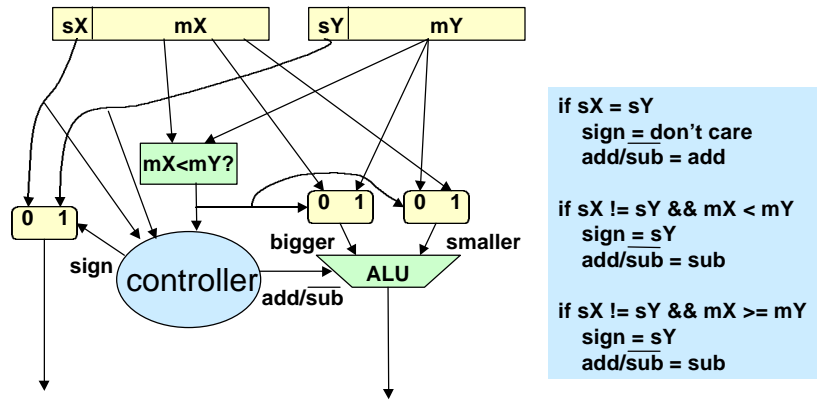
-3

if signs are
same then
check magnitude

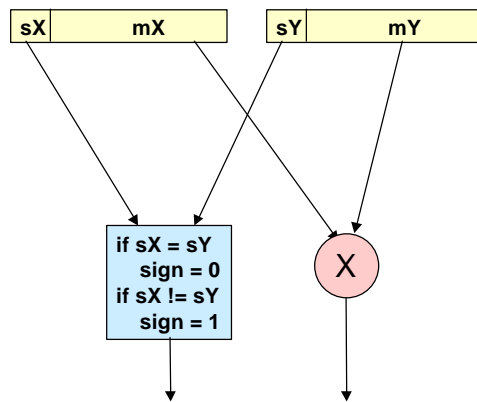


sX	sY	$[X < Y]$
0	0	$[mX < mY]$
0	1	0
1	0	1
1	1	$\sim([mX < mY] + [mX = mY])$

Sign Magnitude: Addition



Sign Magnitude: Multiplication



Fixed point non-integer

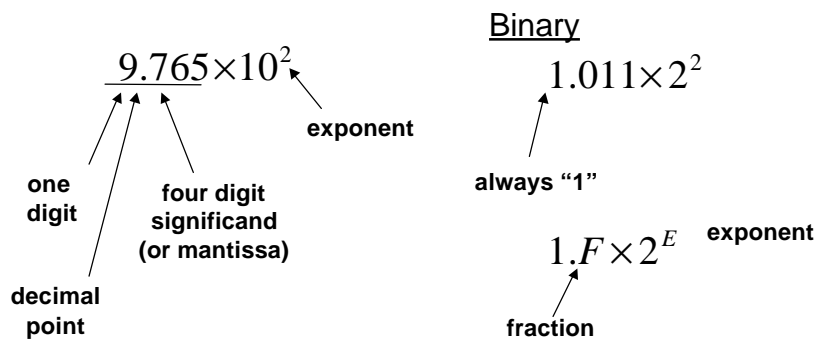
$$b_{n-1}b_{n-2}\dots b_0 \cdot b_{-1}b_{-2}b_{-3}\dots b_{-m} = \sum_{k=-m}^n b_k 2^k$$

binary point

Inconvenient for very small numbers or very large numbers --- lots of digits.

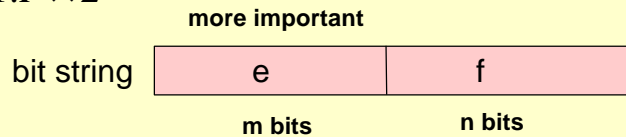
Solution: scientific notation!

Scientific Notation



How do we represent in bits?

$$1.F \times 2^E$$



e = unsigned integer for E $0 \leq E \leq 2^m - 1$
 f = fractional part of significand

What if E can be negative? Use another representation

E.g., choose $bias = 2^{m-1}$

$E = e - bias$ $-bias \leq E \leq 2^m - 1 - bias$

Scientific Notation: Addition

Decimal Example

$$\begin{array}{l} \overbrace{2.780 \times 10^{-1} + 9.765 \times 10^0} \\ \leftarrow \\ \text{4 digits} \end{array}$$

Step 1: Adjust exponent of smaller magnitude #

$$\begin{array}{r} 0.2780 \times 10^0 \\ + 9.765 \times 10^0 \end{array}$$

Step 2: Add significands

$$\begin{array}{r} 0.2780 \times 10^0 \\ + 9.765 \times 10^0 \\ \hline \end{array}$$

$$\textcircled{10.0430} \times 10^0 \quad \text{1 or 2 digits}$$

Step 3: Normalize

$$1.00430 \times 10^1$$

Step 4: Round to 4 digits

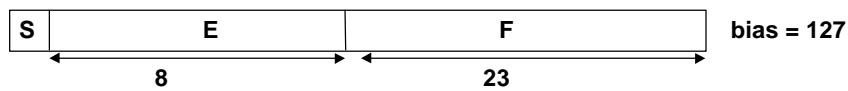
$$1.004 \times 10^1$$

IEEE 754 Standard

$$(-1)^S 1.F \times 2^{E-Bias}$$

hidden bit fraction of significand

IEEE 754 Single Precision (32 bits)



Example

$$-1.10010 \times 2^6$$

S = 1
E = 6+Bias = 133 (decimal) = 10000101
F = 10010

Galen Sasaki

EE 361 University of Hawaii

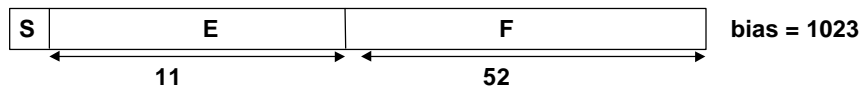
63

IEEE 754 Standard

$$(-1)^S 1.F \times 2^{E-Bias}$$

hidden bit fraction of significand

IEEE 754 Double Precision (64 bits)



Example

$$-1.10010 \times 2^6$$

S = 1
E = 6+Bias = 1029 (decimal) = 10000000101
F = 10010

Galen Sasaki

EE 361 University of Hawaii

64