

CSE 141 – Computer Architecture Spring 2005

Lecture 5 ALU Part 2 April 5, 2005

Pramod V. Argade

CSE141: Introduction to Computer Architecture

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hour:
Mon. 5:00 - 6:00 PM (AP&M 5218)

TAs:

Baris Arslan: barslan@cs.ucsd.edu
Chris Roedel: croedel@cs.ucsd.edu
Raid Ayoub: rayoub@cs.ucsd.edu
Leo Porter: leporter@cs.ucsd.edu

Textbook: Computer Organization & Design
The Hardware Software Interface, 3rd Edition.
Authors: Patterson and Hennessy

Web-page: <http://www.cse.ucsd.edu/classes/sp05/cse141>

Announcements

- **Reading Assignment**

- Chapter 3. Arithmetic for Computers
Sections B6, In More Depth 3.23, 3.4 - 3.8

- **Homework 2: Due Mon., April 18 in class**

3.13, 3.18, 3.19, 3.27, 3.30, 3.35, 3.38

Multiply $-11x-10$ using Booth's algorithm and 5-bit 2-complement representation of multiplicand and multiplier.

- **Quiz**

When: Mon., April 18th, First 10 minutes of the class

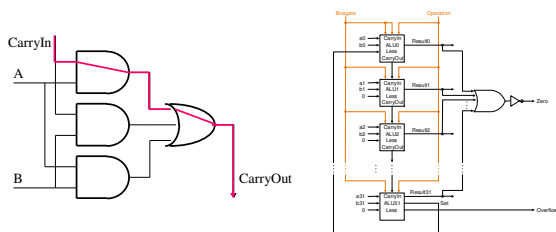
Topic: Arithmetic for Computers, Chapter 3 **Need:** Paper, pen

Course Schedule

Lecture #	Date	Day	Lecture Topic	Quiz Topic	Homework Due
1	3/28	Monday	Introduction, Ch. 1	-	-
2	3/30	Wednesday	Performance, Ch. 4	-	-
3	4/4	Monday	ISA, Ch. 2	Performance	#1
4	4/6	Wednesday	Arithmetic, Ch. 3	-	-
5	4/11	Monday	Arithmetic, Ch. 3	ISA	#2
6	4/13	Wednesday	Single cycle CPU, Ch. 5	-	-
7	4/18	Monday	Single cycle CPU, Ch. 5	Arithmetic	#3
8	4/20	Wednesday	Multi-cycle CPU, Ch. 5	-	-
9	4/25	Monday	Multi-cycle CPU, Ch. 5	Single Cycle CPU	#4
10	4/27	Wednesday	Review for the Midterm	-	-
	5/2	Monday	Mid-term Exam	-	-
11	5/4	Wednesday	Exceptions, Ch. 5 and Pipelining, Ch. 6	-	-
12	5/9	Monday	Pipelining, Ch. 6	-	-
13	5/11	Wednesday	Data and control hazards, Ch. 6	-	-
14	5/16	Monday	Data and control hazards, Ch. 6	Pipeline Hazards	#5
15	5/18	Wednesday	Memory & cache design, Ch. 7	-	-
16	5/23	Monday	Memory & cache design, Ch. 7	Cache	#6
17	5/25	Wednesday	Virtual Memory & cache design, Ch. 7	-	-
No Class	5/30	Monday	Memorial Day Holiday	-	-
18	6/1	Wednesday	Course Review	-	-
	6/10	Friday	Final Exam	-	-

Adder in our ALU is in timing critical path

- The adder we just built is called a "Ripple Carry Adder"
 - The carry bit may have to propagate from LSB to MSB
 - Worst case delay for an N-bit RC adder: 2N-gate delay



- Single gate delay = 0.02 ns (inverter "speed" of 50 GHz)
- 32 bit adder => 64 gate delay => 1.28 ns delay
- Accounting for CLK2Q, set up time and clock skew, the ALU will run at << 789 MHz

Problem: ripple carry adder is slow

- Is there more than one way to do addition?
 - two extremes: ripple carry and sum-of-products

Can you see the ripple? How could you get rid of it?

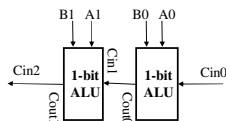
$$c_1 = b_0c_0 + a_0c_0 + a_0b_0$$

$$c_2 = b_1c_1 + a_1c_1 + a_1b_1$$

$$c_3 = b_2c_2 + a_2c_2 + a_2b_2$$

$$c_4 = b_3c_3 + a_3c_3 + a_3b_3$$

The Theory Behind Carry Look-ahead



- Recall: CarryOut = (B & CarryIn) | (A & CarryIn) | (A & B)
 - Cin1 = Cout0 = (B0 & Cin0) | (A0 & Cin0) | (A0 & B0)
 - Cin2 = Cout1 = (B1 & Cin1) | (A1 & Cin1) | (A1 & B1)
- Substituting Cin1 into Cin2:
 - Cin2 = (A1 & A0 & B0) | (A1 & A0 & Cin0) | (A1 & B0 & Cin0) | (B1 & A0 & B0) | (B1 & A0 & Cin0) | (B1 & B0 & Cin0) | (A1 & B1)
- Now define two new terms:
 - Generate** Carry at Bit i $g_i = A_i \& B_i$
 - Propagate** Carry via Bit i $p_i = A_i \mid B_i$
 - Cin1 = Cin0(A0 | B0) | (A0 & B0) = (Cin0 & p0) | g0
 - Cin2 = Cin1(A1 | B1) | (A1 & B1) = (Cin1 & p1) | g1

Carry Lookahead: 1st Level Abstraction

- Using the two new terms we just defined:
 - Generate Carry at Bit i $g_i = A_i \& B_i$
 - Propagate Carry via Bit i $p_i = A_i \mid B_i$
- We can rewrite:
 - Cin1 = g0 | (p0 & Cin0)
 - Cin2 = g1 | (p1 & g0) | (p1 & p0 & Cin0)
 - Cin3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & Cin0)
- Carry going into bit 3 is 1 if
 - We generate a carry at bit 2 (g2)
 - Or we generate a carry at bit 1 (g1) and bit 2 allows it to propagate (p2 & g1)
 - Or we generate a carry at bit 0 (g0) and bit 1 as well as bit 2 allows it to propagate (p2 & p1 & g0)
 - Or we have a carry input at bit 0 (Cin0) and bit 0, 1, and 2 all allow it to propagate (p2 & p1 & p0 & Cin0)

Carry Lookahead: 2nd Level of Abstraction

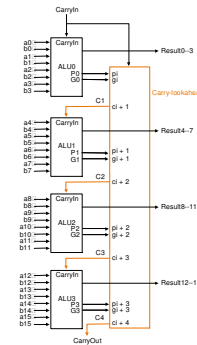
- Propagate signals for 4-bit adders (1 gate delay to combine p's)
 - $P0 = p3.p2.p1.p0$
 - $P1 = p7.p6.p5.p4$
 - $P2 = p11.p10.p9.p8$
 - $P3 = p15.p14.p13.p12$
- Generate signals for 4-bit adders (2 gate delays to combine p's and g's)
 - $G0 = g3 + (p3.g2) + (p3.p2.g1) + (p3.p2.p1.g0)$
 - $G1 = g7 + (p7.g6) + (p7.p6.g5) + (p7.p6.p5.g4)$
 - $G2 = g11 + (p11.g10) + (p11.p10.g9) + (p11.p10.p9.g8)$
 - $G3 = g15 + (p15.g14) + (p15.p14.g13) + (p15.p14.p13.g12)$
- 4-bit adder carry computation (2 gate delays to combine G's P's and c0)
 - $C1 = G0 + (P0.c0)$
 - $C2 = G1 + (P1.G0) + (P1.P0.c0)$
 - $C3 = G2 + (P2.G1) + (P2.P1.G0) + (P2.P1.P0.c0)$
 - $C4 = G3 + (P3.G2) + (P3.P2.G1) + (P3.P2.P1.G0) + (P3.P2.P1.P0.c0)$
- Total $2 + 2 + 1 = 5$ levels of logic to compute c16
 - C4 has 2 levels of logic with Gi, Pi
 - Gi has 2 levels of logic with gi, pi
 - gi & pi have 1 level of logic with inputs Ai, Bi

Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-9

16-bit Adder from Four 4-bit ALUs



- 16-bit Ripple Carry (RC) adder has $16 \times 2 = 32$ gate delays
- 16-bit Carry-Look-Ahead adder (CLA) has 5 gate delays
- CLA adder is faster than RC by a factor of $32/5 \approx 6$

Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-10

Grade school Multiplication algorithm

- In general (ignoring sign bits):
 - m bits \times n bits = $(m+n)$ bit product
- Binary makes it easy:
 - 0 => place 0 (0 x multiplicand)
 - 1 => place multiplicand (1 x multiplicand)
- Paper and pencil example of binary multiplication:

$(8 \times 10 = 80, 0x8 \times 0xa = 0x50)$

```

      1000 (multiplicand)
      x 1010 (multiplier)
      -----
      0000
      1000x
      0000xx
      1000xxx
      -----
      1010000 (Result)
    
```

Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-11

Observations about Multiplication

- More complicated than addition
- Simple algorithm:
 - Accomplished via shift and add
- More time delay and more gates (=> silicon area)
- Let's look at 3 versions based on grade school algorithm

Pramod Argade

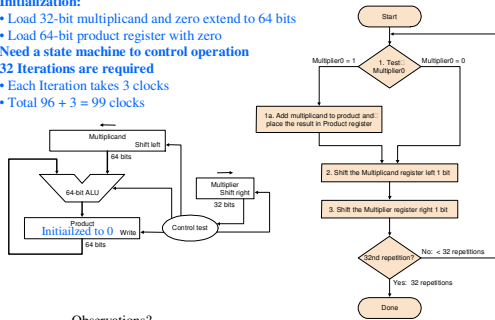
UCSD CSE 141, Spring 2005

Slide 5-12

Multiplication: First Version

Initialization:

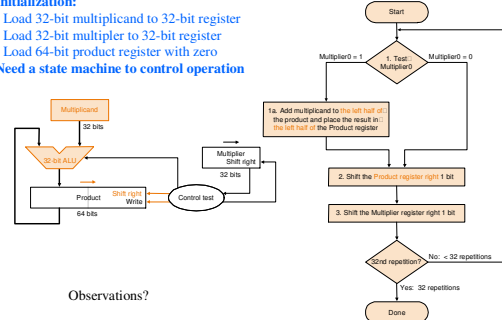
- Load 32-bit multiplicand and zero extend to 64 bits
- Load 64-bit product register with zero
- Need a state machine to control operation
- 32 Iterations are required
- Each Iteration takes 3 clocks
- Total $96 + 3 = 99$ clocks



Multiplication: Second Version

Initialization:

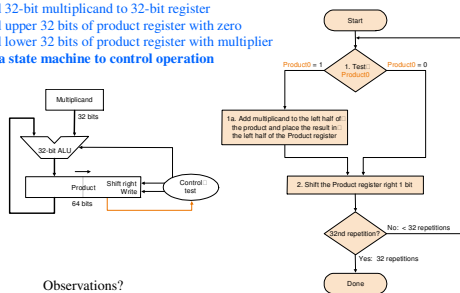
- Load 32-bit multiplicand to 32-bit register
- Load 32-bit multiplier to 32-bit register
- Load 64-bit product register with zero
- Need a state machine to control operation



Multiplication: Third Version

Initialization:

- Load 32-bit multiplicand to 32-bit register
- Load upper 32 bits of product register with zero
- Load lower 32 bits of product register with multiplier
- Need a state machine to control operation



Multiplying Signed Numbers

- Convert all operands to positive
- Determine sign of the product
 - Sign of the product = $\text{sign}(op1) \wedge \text{sign}(op2)$
- Multiply positive operands (only 31 bits)
- If the sign of the result is negative, negate the result
- Previous approach will work:
 - Must extend sign of the product when shifting

Is there a better way?

Booth's Algorithm

- An elegant approach to multiplying signed numbers
- With ability to add, subtract and shift
 - There are multiple ways to do multiply
- Consider signed operands A and B

$$\begin{aligned}
 A &= (A_{31} \cdot 2^{31}) + (A_{30} \cdot 2^{30}) + (A_{29} \cdot 2^{29}) + \dots + (A_1 \cdot 2^1) + (A_0 \cdot 2^0) \\
 &= (-A_{31} \cdot 2^{31}) + (2A_{30} - A_{30}) \cdot 2^{30} + (2A_{29} - A_{29}) \cdot 2^{29} + \dots + (2A_0 - A_0) \cdot 2^0 \\
 &= (A_{30} - A_{31}) \cdot 2^{31} + (A_{29} - A_{30}) \cdot 2^{30} + \dots + (A_1 - A_2) \cdot 2^1 + (A_1 - A_0) \cdot 2^0 \\
 A \cdot B &= [(A_{30} - A_{31}) \cdot 2^{31} + (A_{29} - A_{30}) \cdot 2^{30} + \dots + (A_1 - A_2) \cdot 2^1 + (A_1 - A_0) \cdot 2^0] \cdot B \\
 &= (A_{30} - A_{31}) \cdot 2^{31} \cdot B + (A_{29} - A_{30}) \cdot 2^{30} \cdot B + \dots + (A_1 - A_2) \cdot 2^1 \cdot B + (A_1 - A_0) \cdot 2^0 \cdot B
 \end{aligned}$$

- Recipe:

Evaluate $(A_{i-1} - A_i)$

0: Do nothing

1: Add B

-1: Subtract B

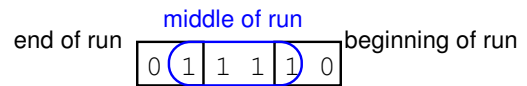
Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-17

Booths algorithm: Signed multiplication

$$A \cdot B = (A_{30} - A_{31}) \cdot 2^{31} \cdot B + (A_{29} - A_{30}) \cdot 2^{30} \cdot B + \dots + (A_1 - A_2) \cdot 2^1 \cdot B + (A_1 - A_0) \cdot 2^0 \cdot B$$



Current Bit	Bit to the Right	Explanation	Example	Op
1	0	Begins run of 1s	000111 <u>0</u> 00	sub
1	1	Middle of run of 1s	000111 <u>1</u> 000	none
0	1	End of run of 1s	000 <u>1</u> 111000	add
0	0	Middle of run of 0s	0001111000	none

Originally for Speed (when shift was faster than add)

- Replace a string of 1s in multiplier with an initial subtract when we first see a one and then later add for the bit after the last one

Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-18

Booth's Algorithm

- Recipe: for $A \cdot B$

Add $A_{i-1} = 0$

Evaluate $(A_{i-1} - A_i)$

0: Do nothing

1: Add B

-1: Subtract B

- Example: Use Booth's Algorithm for following multiplication

$$2 * (-6) = 0010 * 1010 = -12 = 1111 0100$$

Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-19

Division

$$\begin{array}{r}
 \text{Divisor } 1000 \quad \begin{array}{l} 1001 \text{ Quotient} \\ 1001010 \text{ Dividend} \\ \hline -1000 \\ \hline 10 \\ 101 \\ \hline 1010 \\ \hline -1000 \\ \hline 10 \end{array} \\
 \hline
 \end{array}$$

Remainder (or Modulo result)

See how big a number can be subtracted, creating quotient bit on each step

Binary $\Rightarrow 1 * \text{divisor}$ or $0 * \text{divisor}$

Dividend = Quotient x Divisor + Remainder

$\Rightarrow \text{sizeof}(\text{Dividend}) = \text{sizeof}(\text{Quotient}) + \text{sizeof}(\text{Divisor})$

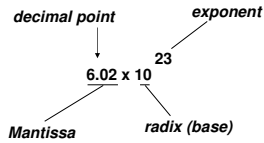
3 versions of divide, successive refinement

Pramod Argade

UCSD CSE 141, Spring 2005

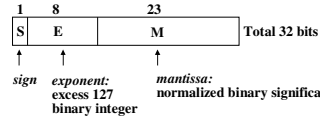
Slide 5-20

Recall Scientific Notation



IEEE Single Precision F.P. $\pm 1.M \times 2^{E-127}$

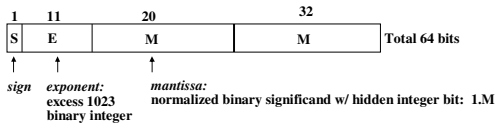
IEEE 754 Single-precision Floating-Point



$$N = (-1)^S (1.M) 2^{E-127}$$

- **Example:**
Convert - 325.75 to IEEE Single Precision Floating Point Representation

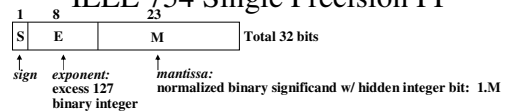
IEEE 754 Double-precision Floating-Point



$$N = (-1)^S (1.M) 2^{E-1023}$$

- **Example:**
Convert - 325.75 to IEEE Double Precision Floating Point Representation

IEEE 754 Single Precision FP

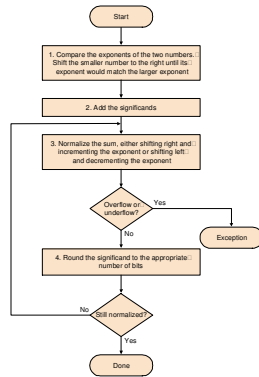


- If E=255 and F is nonzero, then V = NaN ("Not a number")
- If E=255 and F is zero and S is 1, then V = -Infinity
- If E=255 and F is zero and S is 0, then V = Infinity
- If 0 < E < 255 then V = $(-1)^S \times 2^{(E-127)} \times (1.F)$
- If E=0 and F is zero and S is 1, then V = -0
- If E=0 and F is zero and S is 0, then V = 0

In particular,

- 0 00000000 000000000000000000000000 = 0
- 1 00000000 000000000000000000000000 = -0
- 0 11111111 000000000000000000000000 = Infinity
- 1 11111111 000000000000000000000000 = -Infinity
- 0 11111111 000001000000000000000000 = NaN
- 1 11111111 001000100010001010101010 = NaN
- 0 10000000 000000000000000000000000 = $+1 \times 2^{*(128-127)} \times 1.0 = 2$

Floating Point Addition



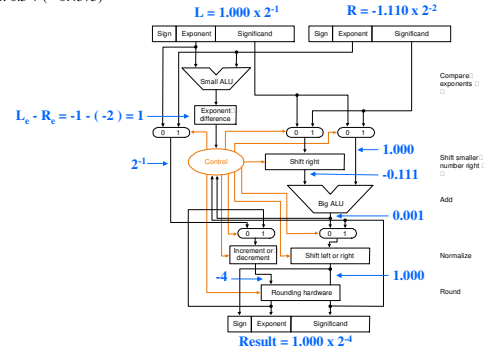
Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-29

Floating Point Addition

Example: $0.5 + (-0.4375)$



Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-30

IEEE 754 Floating Point

- Increasing the size of significand enhances accuracy
- Increasing the size of exponent increases the range of the numbers that can be represented
- Special representation of 0 (E = 00000000)
- Overflow or underflow can happen
- Can do integer compare for greater-than, sign:
- Single Precision
 - Range of about 2×10^{-38} to 2×10^{38}
- Double Precision
 - Range of about 2×10^{-308} to 2×10^{308}
- Infinite variety of real numbers exist between, say, 0 and 1
 - Not more than 2^{53} can be represented exactly in double precision

Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-31

Floating Point Complexities

- Operations are somewhat more complicated
- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
 - IEEE 754 keeps two extra bits, guard and round
 - four rounding modes
 - positive divided by zero yields “infinity”
 - zero divide by zero yields “not a number”
- Implementing the standard can be tricky
- Not using the standard can be even worse
 - See text for description of 80x86 and Pentium bug!

Pramod Argade

UCSD CSE 141, Spring 2005

Slide 5-32

Summary

- Multiplication and division take much longer than addition, requiring multiple addition steps.
- Floating Point extends the range of numbers that can be represented, at the expense of precision (accuracy).
- FP operations are very similar to integer, but with pre- and post-processing.

Announcements

- **Reading Assignment**
 - Chapter 3. Arithmetic for Computers
Sections B6, In More Depth 3.23, 3.4 - 3.8
- **Homework 2: Due Mon., April 18 in class**
3.13, 3.18, 3.19, 3.27, 3.30, 3.35, 3.38
Multiply -11×-10 using Booth's algorithm and 5-bit 2-complement representation of multiplicand and multiplier.
- **Quiz**
When: Mon., April 18th, First 10 minutes of the class
Topic: Arithmetic for Computers, Chapter 3 **Need:** Paper, pen