Nadar Saraswathi College of Engineering & Technology

NSCET E-LEARNING PRESENTATION

LISTEN … LEARN… LEAD…

Computer Science Engineering

II YEAR / IV SEMESTER

CS8491-Computer Architecture

B.Sri Chithra Devi,M.E
Assistant Professor
Nadar Saraswathi College of Engineering & Technology,
Vadapudupatti, Annanji (po), Theni – 625531.

UNIT-II

ARITHMETIC FOR COMPUTERS

Arithmetic for Computers

- Addition and Subtraction
- Multiplication
- Division
- Floating Point Representation
- Floating Point Operations
- Subword Parallelism

Arithmetic for Computers

Operations on integers

- Addition and subtraction
- Multiplication and division
- Dealing with overflow

Floating-point real numbers

Representation and operations

Lecture1-Introduction

Operations on integers

- Addition and subtraction
- Multiplication and division
- Dealing with overflow

Floating-point real numbers

Representation and operations

MIPS Arithmetic Logic Unit (ALU)

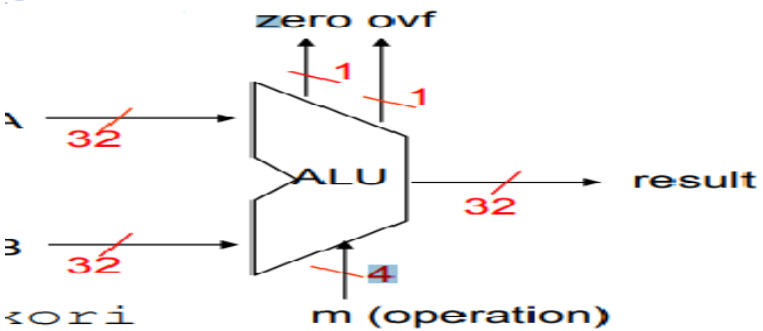
Must support the Arithmetic/Logic operations of the ISA

add, addi, addiu, addu
sub, subu
mult, multu,
div, divu
Sqrt
and, andi, nor, or, ori, xor, xori
beq, bne, slt, slti, sltiu, sltu

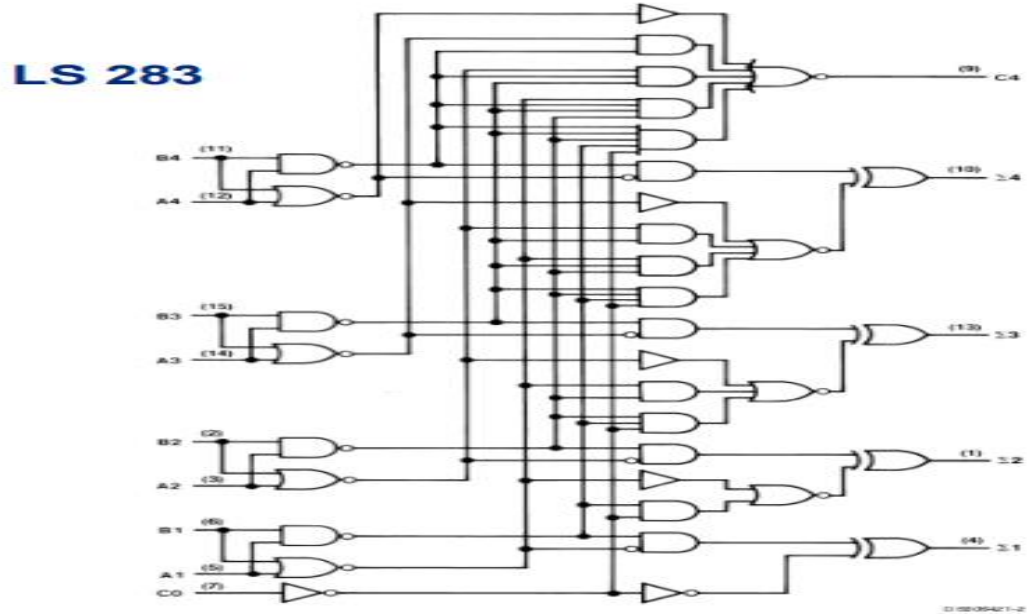
MIPS Arithmetic Logic Unit (ALU)

With special handling for

- **sign extend** – addi, addiu, slti, sltiu
- **zero extend** – andi, ori, xori
- **overflow detection** – add, addi, sub

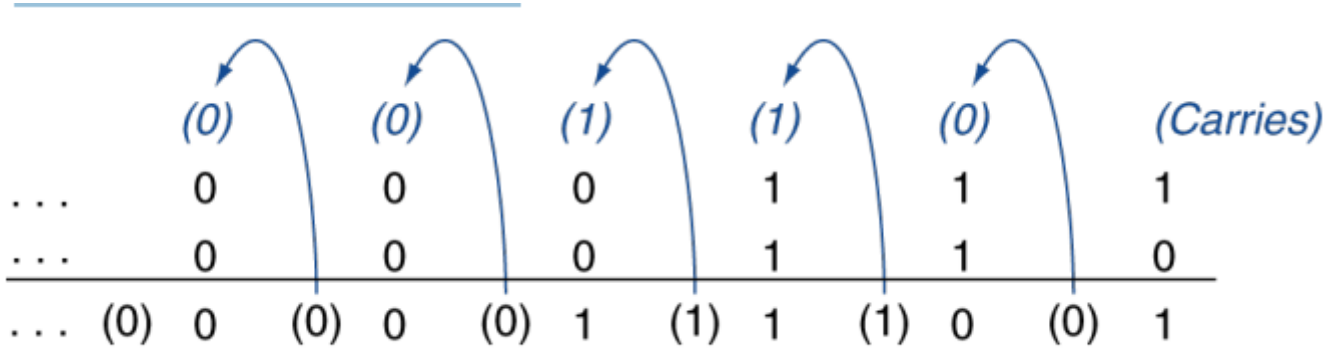


Carry Lookahead Logic (4 bit adder)



Lecture 2-Integer Addition

To perform addition for 7,6



Integer Addition

Overflow if result out of range

- Adding +ve and –ve operands, no overflow
- Adding two +ve operands
- Overflow if result sign is 1
- Adding two –ve operands
- Overflow if result sign is 0

Integer Subtraction

To perform subtraction for+7,-6

+7:	0000 0000 ... 0000 0111
–6:	1111 1111 ... 1111 1010
+1:	0000 0000 ... 0000 0001

Integer Subtraction

Overflow if result out of range

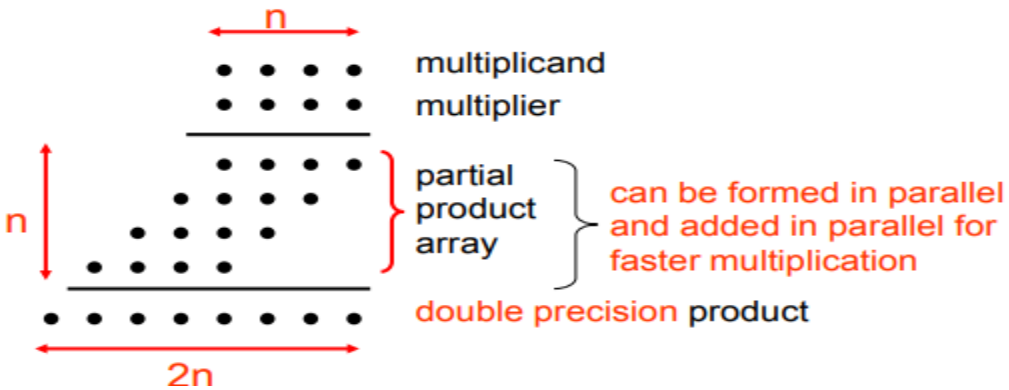
- Subtracting two +ve or two –ve operands, no overflow
  Subtracting +ve from –ve operand

Overflow if result sign is 0

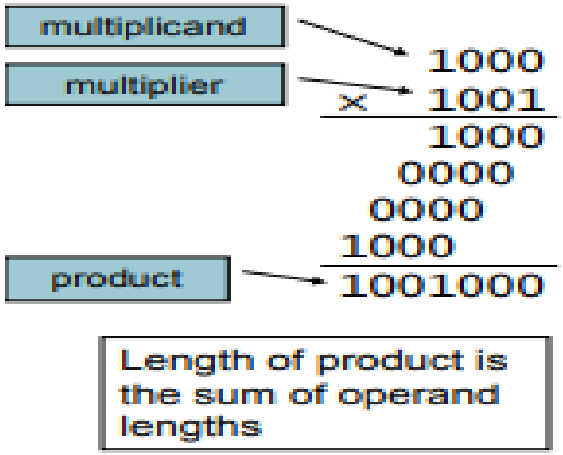
- Subtracting –ve from +ve operand
- Overflow if result sign is 1

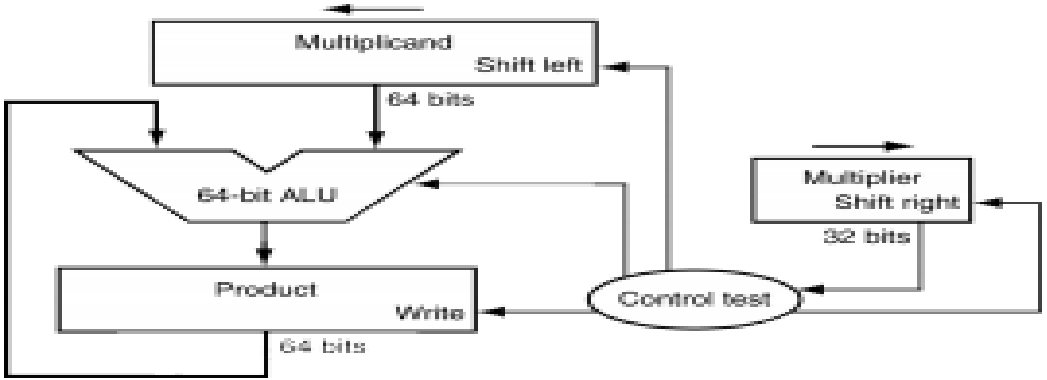
Lecture 3-Multiply

Binary multiplication is just a bunch of right shifts and adds

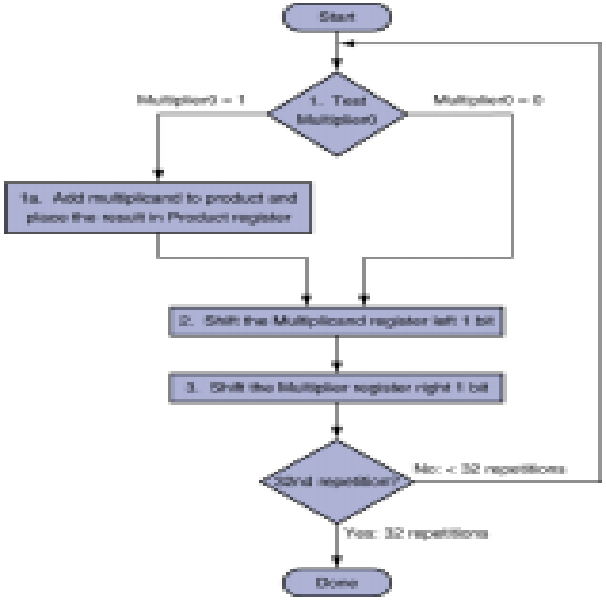


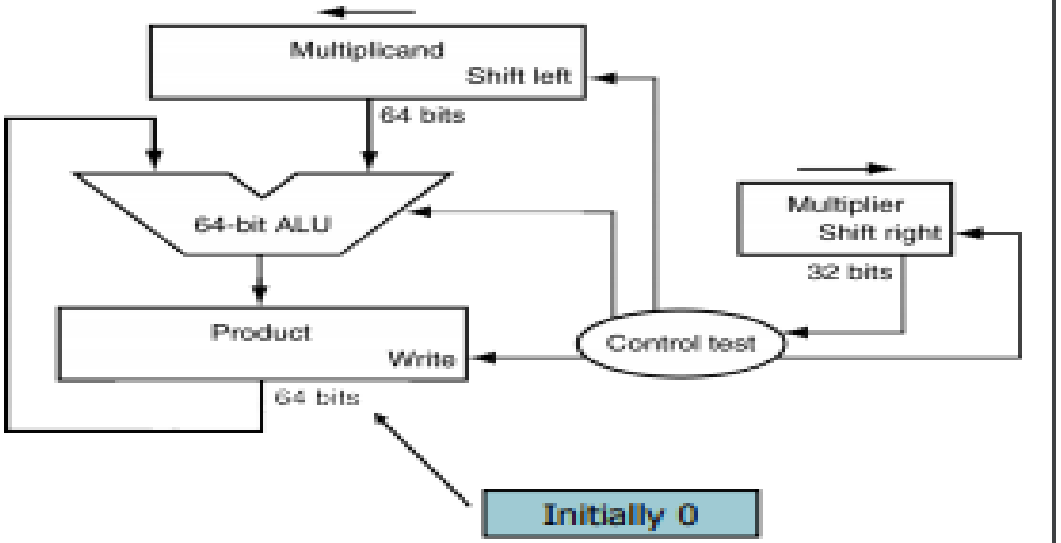
Long-multiplication Approach





Multiplication Hardware





Nadar Saraswathi College of Engineering & Technology

Steps for multiplication:

2 x 3 or 0010 x 0011

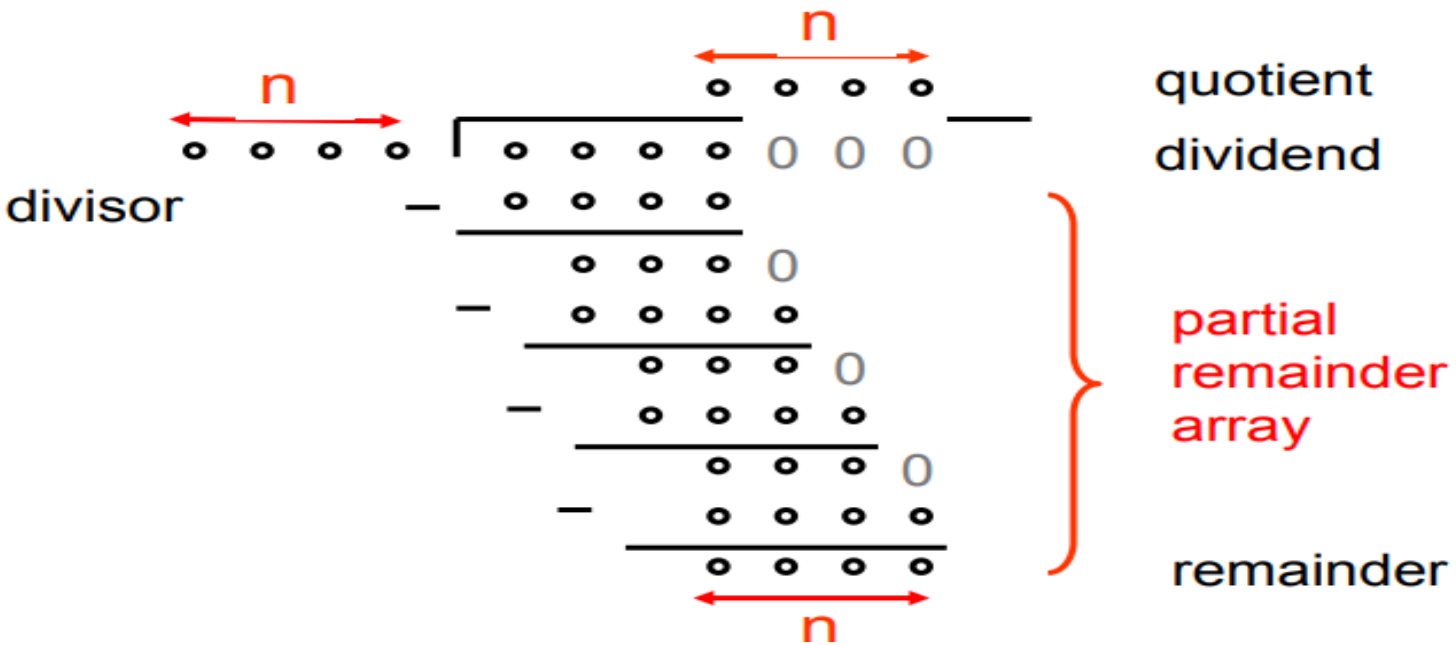
Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⟹ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⟹ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	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

Lecture 4-Division

- Division is just a bunch of quotient digit guesses and left shifts and subtracts

dividend = quotient x divisor + remainder

Lecture 4-Division

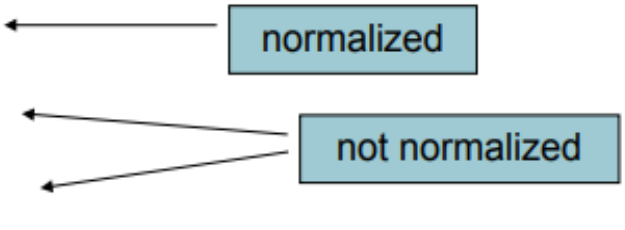


Division Steps

- Check for 0 divisor
- Long division approach
- If divisor ≤ dividend bits
- 1 bit in quotient, subtract
- Otherwise
- 0 bit in quotient, bring down next dividend bit
- Restoring division
- Do the subtract, and if remainder goes < 0, add divisor back Signed division
- Divide using absolute values
- Adjust sign of quotient and remainder as required

Lecture 5-Floating Point Representation

- Representation for non-integral numbers
- Including very small and very large numbers
- Like scientific notation
  - -2.34×10^{56}
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^{9}$



Floating Point Representation

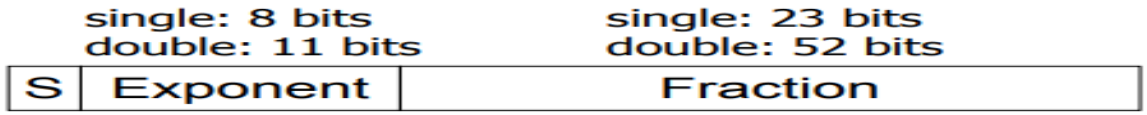
- The **floating number representation** of a **number** has two part: the first part represents a signed fixed **point number** called mantissa.
- The second part of designates the position of the decimal (or binary) **point** and is called the exponent.

Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of representations
- Portability issues for scientific code
- Now almost universally adopted
- Two representations
- Single precision (32-bit)
- Double precision (64-bit)

IEEE Floating-Point Format

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalize significant: 1.0 ≤ |significant| < 2.0
- Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
- Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
- Ensures exponent is unsigned (non-negative)
- Single: Bias = 127; Double: Bias = 1203



$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

IEEE Floating-Point Format

- Single-Precision Range
- Exponents 00000000 and 11111111 reserved
- Smallest value
- Exponent: 00000001 ⇒ actual exponent = 1 – 127 = –126
- Fraction: 000…00 ⇒
- significant = 1.0 ±1.0 × 2–126 ≈ ±1.2 × 10–38
- Largest value exponent: 11111110
- ⇒ actual exponent = 254 – 127 = +127
- Fraction: 111…11 ⇒ significant ≈ 2.0
- ±2.0 × 2+127 ≈ ±3.4 × 10+38 Powers of Ten: Quark 10-16
- Universe 1025

Double-Precision Range

- Exponents 0000…00 and 1111…11 reserved

Smallest value

- Exponent: 00000000001
- actual exponent = 1 – 1023 = –1022
- Fraction: 000…00 ⇒ significant = 1.0
- ±1.0 × 2–1022 ≈ ±2.2 × 10–308

Largest value

- Exponent: 11111111110
- ⇒ actual exponent = 2046 – 1023 = +1023
- Fraction: 111…11 ⇒ significand ≈ 2.0
- ±2.0 × 2+1023 ≈ ±1.8 × 10+308

Floating-Point Precision

- Relative precision
- all fraction bits are significant
- Single: approx 2–23
- Equivalent to 23 × log102 ≈ 23 × 0.3 ≈ 6 decimal digits of precision
- Double: approx 2–52
- Equivalent to 52 × log102 ≈ 52 × 0.3 ≈ 16 decimal digits of precision
- 0.099999999403953552 < 0.1 < 0.1000000149011612

Floating-Point Example

- Represent –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000...00_2$
  - Exponent = –1 + Bias
    - Single: –1 + 127 = 126 = 01111110_2
    - Double: –1 + 1023 = 1022 = 01111111110_2
- Single: 10111111101000...00
- Double: 10111111111101000...00

Lecture 6-Floating-Point operation

Floating-Point Addition

- Consider a 4-digit decimal example
  - $9.999 \times 10^{1} + 1.610 \times 10^{-1}$
- 1. Align decimal points
  - Shift number with smaller exponent
  - $9.999 \times 10^{1} + 0.016 \times 10^{1}$
- 2. Add significands
  - $9.999 \times 10^{1} + 0.016 \times 10^{1} = 10.015 \times 10^{1}$
- 3. Normalize result & check for over/underflow
  - 1.0015×10^{2}
- 4. Round and renormalize if necessary
  - 1.002×10^{2}

FP Adder Hardware

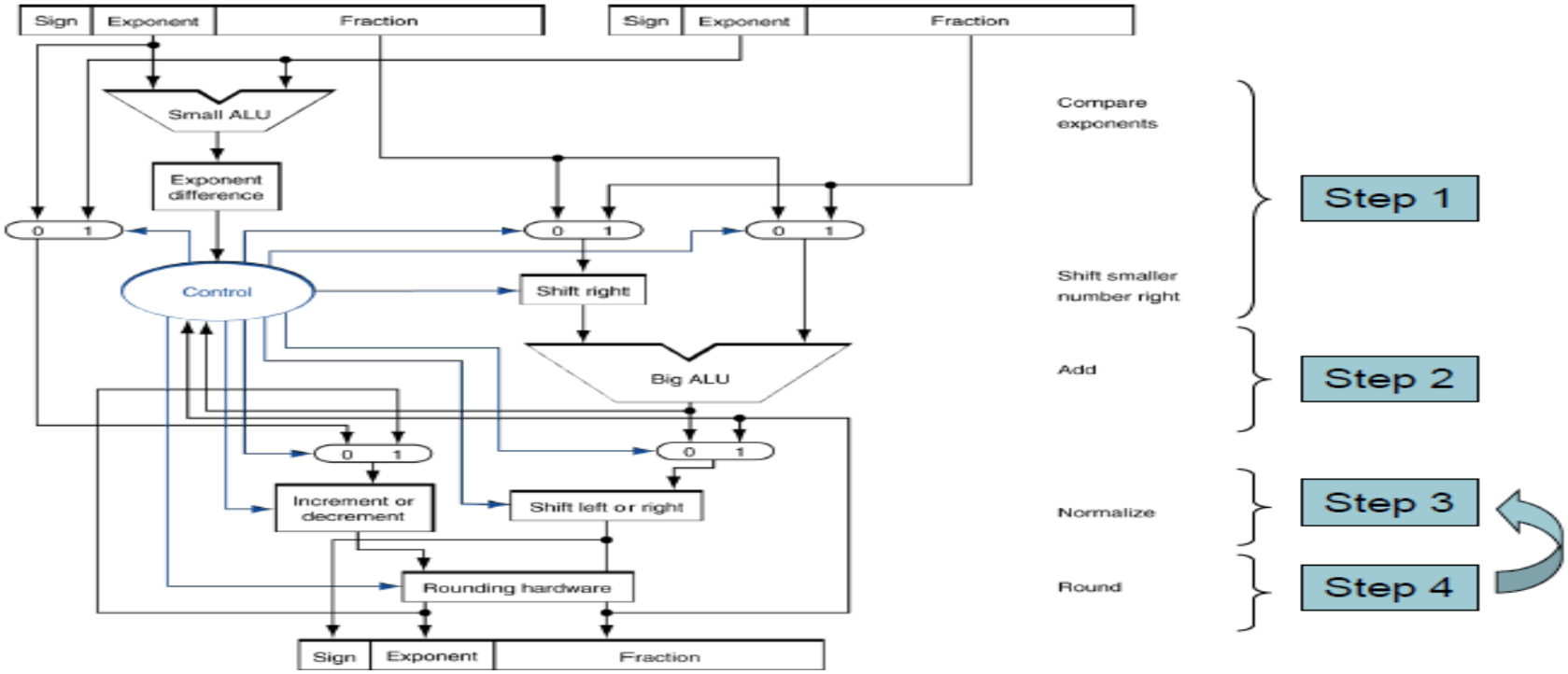
- Much more complex than integer adder.
- Doing it in one clock cycle would take too.
- FP cycles usually takes several cycles.
- It can be pipelined.

Example

Floating-Point Addition

- Now consider a 4-digit binary example
  - $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)
- 1. Align binary points
  - Shift number with smaller exponent
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- 2. Add significands
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- 3. Normalize result & check for over/underflow
  - $1.000_2 \times 2^{-4}$, with no over/underflow
- 4. Round and renormalize if necessary
  - $1.000_2 \times 2^{-4}$ (no change) = 0.0625

FP Adder Hardware



Interpretation of Data

- Bits have no inherent meaning.
- Interpretation depends on the instructions applied.
- Computer representations of numbers.
- Finite range and precision.
- Need to account for this in programs.

Associativity

- Parallel programs may interleave operations in unexpected orders.
- Assumptions of associativity may fail.

		(x+y)+z	x+(y+z)
x	-1.50E+38		-1.50E+38
y	1.50E+38	0.00E+00	
z	1.0	1.0	1.50E+38
		1.00E+00	0.00E+00

x86 FP Architecture

- Originally based on 8087 FP coprocessor.
- 8 × 80-bit extended-precision registers.
- Used as a push-down stack.
- Registers indexed from TOS: ST(0), ST(1).
- FP values are 32-bit or 64 in memory.
- Converted on load/store of memory operand.
- Integer operands can also be converted on load/store.
- Very difficult to generate and optimize code.
- Result: poor FP performance

Streaming SIMD Extension 2 (SSE2)

- Adds 4 × 128-bit registers.
- Extended to 8 registers in AMD64/EM64T.
- Can be used for multiple FP operands.
- 2 × 64-bit double precision.
- 4 × 32-bit double precision.
- Instructions operate on them simultaneously Single-Instruction Multiple-Data

Right Shift and Division

- Left shift by i places multiplies an integer by 2^i
- Right shift divides by 2^i? *Only for unsigned integers*

- For signed integers Arithmetic right shift: replicate the sign bit
  - e.g., –5 / 4 $11111011_2 >> 2 = 11111110_2 = -2$
  - Rounds toward –∞

- c.f. $11111011_2 >>> 2 = 00111110_2 = +62$

Lecture 6-Subword Parallelism

- Given that the parallelism occurs within a wide word, the extensions are classified as subword parallelism.
- It is also classified under the more general name of data level parallelism.
- They have been also called vector or SIMD, for single instruction, multiple data .
- The rising popularity of multimedia applications led to arithmetic instructions that support narrower operations that can easily operate in parallel.
- For example, ARM added more than 100 instructions in the NEON multimedia instruction extension to support subword parallelism, which can be used either with ARMv7 or ARMv8.