SECR2033
Computer Organization and Architecture

Module 2
Data Representation in Computer Systems

Objectives:

- To understand the fundamentals of **numerical data** representation and manipulation in digital computers.
- To master the skill of converting between various **radix systems**.
- To understand how **errors** can occur in computations because of overflow and truncation.
- To understand the fundamental concepts of **floating-point** representation.

Module 2

Data Representation in Computer Systems

Module 2

Data Representation in Computer Systems

2.1 Introduction

- The Arithmetic and Logic Unit

2.2 Fixed-Number (Integer) Re
2.3 Fixed-Number (Integer) Ari
2.4 Floating-Points Representa
2.5 Floating-Points Arithmetic
2.6 Summary

2.1 Introduction

2

Numbers are represented by binary bits:

- *How are negative numbers represented?*
- *What is the largest number that can be represented in a computer world?*
- *What happens if an operation creates a number bigger than can be represented?*
- *What about fractions and real numbers?*
- *A mystery: How does hardware really multiply or divide numbers?*

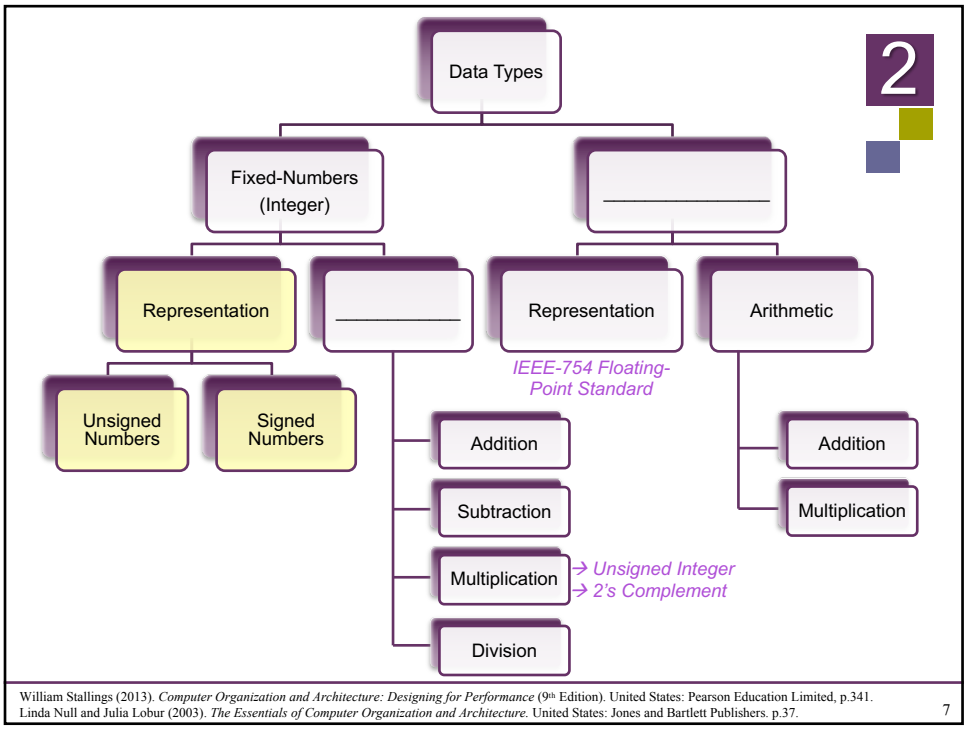
5

2

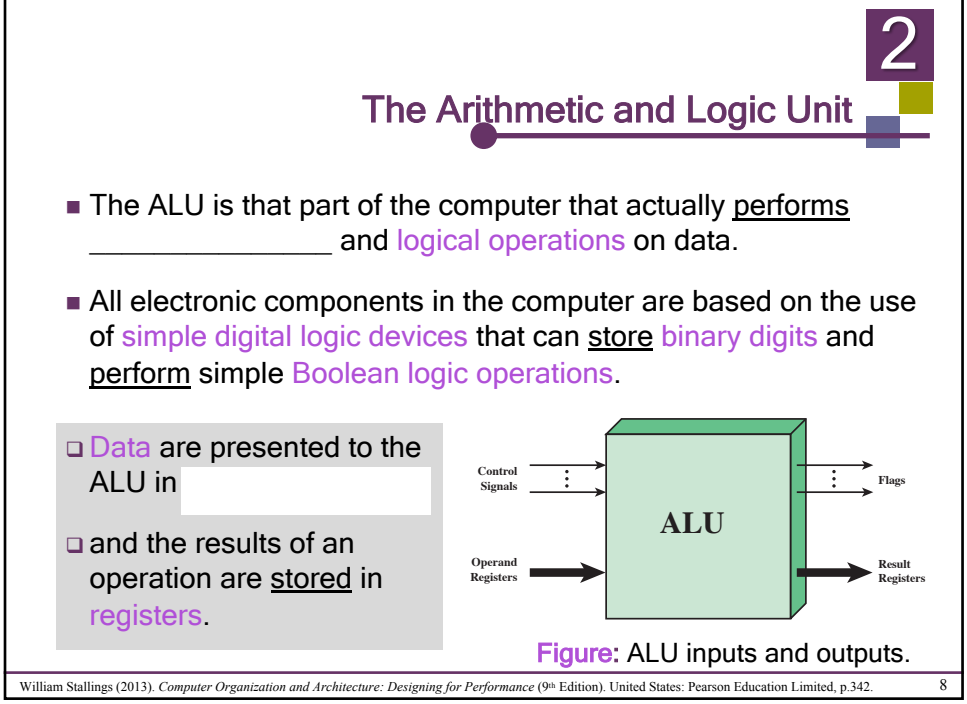
- We begin our examination of the processor with an overview of the arithmetic and logic unit (ALU) → *computer arithmetic*.
- Computer arithmetic is commonly performed on two very different types of numbers: _____ and _____.

- In both cases, the representation chosen is a crucial design issue and is treated first, followed by a discussion of arithmetic operations.

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.342. 6



William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.341.
Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.37. 7



William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.342. 8

Module 2

Data Representation in Computer Systems

RECAP

2

Overview

(Binary Number)

- Numbers are kept in computer hardware as a series of high and low electronic signals.
- They are considered base 2 numbers (_____)
- All information is composed of binary digits or *bits*.
- In any number base, the value of i th digit d is

$$d \times Base^{i}$$

where i start at 0 and increases from right to left.

10

RECAP

2

Binary Equivalents

- 1 *Nybble* (or *nibble*) = _____ *bits*
- 1 *Byte* = 2 *nybbles* = _____ *bits*
- 1 *Kiloyte* (*KB*) = _____ *Bytes*
- 1 *Megabyte* (*MB*) = 1024 *Kilobytes* = 1,048,576 *Bytes*
- 1 *Gigabyte* (*GB*) = 1024 *Megabytes* = 1,073,741,824 *Bytes*

11

RECAP

2

Overview

(Sign Extension)

- Extending a number representation to a larger number of bits.
- **Example**: 2 in 8 bit binary to 16 bit binary.

00000010 → 00000000 00000010

- In signed numbers, it is important to extend the sign bit to *preserve* the number (+ve or -ve)
- **Example**: (–2) in 8 bit binary to 16 bit binary.

11111110 → 11111111 11111110

Sign bit *Sign bit extended* *Sign bit*

12

RECAP

2

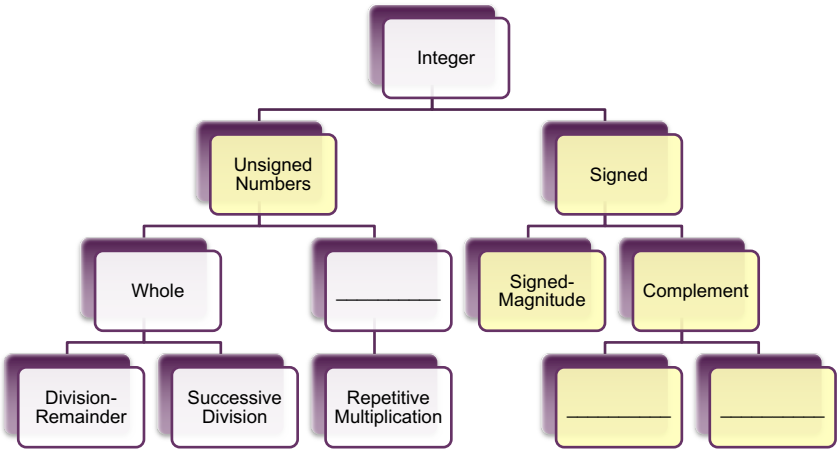


Figure: Types of numbers.

13

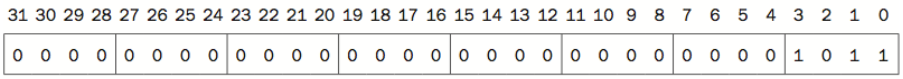
RECAP

2

Unsigned Numbers

MIPS (Millions Instruction Per Second)
LSB (Least Significant Bit)
MSB (Most Significant Bit)

- Since MIPS word is 32 bits long:
  - LSB → bit _____
  - MSB → bit _____



(32 bits wide)

Figure: The numbering of bits in MIPS word for 1011_2

- Range Number can represent : 0 to $2^{32} - 1$

  $0 - 4{,}294{,}967{,}295_{10}$

14

RECAP

2

Signed Numbers
(a) Signed-Magnitude

- The conversions we have so far presented have involved only positive numbers.
- To represent negative values, computer systems allocate the high-order bit to indicate the sign of a value.
  - The high-order bit is the leftmost bit in a byte. It is also called the _____.
- The remaining bits contain the value of the number.

15

RECAP

2

Example 1:

Add 79_{10} to 35_{10} using signed-magnitude arithmetic in 8-bit binary.

			1	1	1	1		⇐ carries
0		1	0	0	1	1	1	(79)
0	+	0	1	0	0	0	1	1	+ (35)
0		1	1	1	0	0	1	0	(114)

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture*. United States: Jones and Bartlett Publishers. p.45.

16

RECAP

2

- The sum of two positive numbers, which is positive.
- *Overflow* (and thus an erroneous result) in signed numbers occurs when the sign of the result is incorrect.
- The _____ is used only for the sign, so we can't "carry into" it, otherwise the result will be truncated as the MSB bit overflows, giving an incorrect sum.

- If the *overflow* bit is not discard, it would carry into the sign, causing the more outrageous result of the sum of two positive numbers being negative.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.45-46. 17

RECAP

2

Example 2:

Add 01001111_2 to 01100011_2 using signed-magnitude arithmetic in 8-bit binary.

Last carry	1 ←			1	1	1	1		⇐ carries
overflows and is	0		1	0	0	1	1	1	1 (79)
discarded.	0	+	1	1	0	0	0	1	1 + (99)
	0		0	1	1	0	0	1	0 (50)

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.46. 18

RECAP

2

Example 3:

Add (-19_{10}) to 13_{10} using signed-magnitude arithmetic in 8-bit binary.

- The first number (_____) is negative (sign bit 1)
- The second number (_____) is positive (sign bit 0)
- Since larger magnitude is *augend*, then subtract to *addend*.

			0	1	2			⇐ borrows	
1		0	0	~~1~~	~~0~~	~~0~~	1	1	(−19)
0	−	0	0	0	1	1	0	1	+ (13)
1		0	0	0	0	1	1	0	(−6)

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.48. 19

RECAP

2

- In signed magnitude, the sign bit is used only for the sign.

Problem 1:

If there any carry emitting from the last bit, the result will be truncated as the last bit overflow, giving an incorrect sum.

Problem 2:

Complicated to define the larger magnitude, to subtract negative number, borrow from the *minuend*.

Solution: Need a simpler method for representing signed numbers → _____.

20

RECAP

2

Signed Numbers
(b) One's Complement

- In complement systems, _____ values are represented by some difference between a number and its base.
- In *diminished radix complement* systems, a negative value is given by the difference between the absolute value of a number and one less than its base.
- In the binary system, this gives us one's complement.
  - It amounts to little more than flipping the bits of a binary number.

21

RECAP

2

Example 4:

Using one's complement 8-bit binary arithmetic, find the sum of 9_{10} and (-23_{10}).

The last	$0 \leftarrow 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1$	(9)
carry is zero	$+\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0$	+ (–23)
so we are done.	$1\ 1\ 1\ 1\ 0\ 0\ 0\ 1$	-14_{10}

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.51.

22

RECAP

2

Example 5:

Using one's complement 8-bit binary arithmetic, find the sum of (-9_{10}) and 23_{10}.

	1	←	1	1	1		1	1	⇐ carries
			0	0	0	1	0	1	1 1 (23)
The last		+	1	1	1	1	0	1	1 0 +(–9)
carry is added			0	0	0	0	1	1	0 1
								+	1
to the sum.			0	0	0	0	1	1	1 0 14_{10}

With one's complement addition, the carry bit is "carried around" and added to the sum.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.51. 23

RECAP

2

Signed Numbers

(c) Two's Complement

- Two's complement is an example of a *radix complement*.
- The radix complement is often more intuitive than the *diminished radix complement*.
- The two's complement is nothing more than one's complement incremented by 1.
- To find the two's complement of a binary number, simply _____ bits and _____.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.51. 24

RECAP

2

Example 6:

Express 23_{10}, (-23_{10}), and (-9_{10}) in 8-bit binary two's complement form.

$$23_{10} =$$
$$-23_{10} =$$
$$-9_{10} =$$

Example 7:

Add 9_{10} to (-23_{10}) using 8-bit binary two's complement arithmetic.

$$\begin{array}{rr} 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1 & (9) \\ +\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1 & +(-23) \\ \hline 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0 & -14_{10} \end{array}$$

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.52. 25

RECAP

2

Example 8:

Find the sum of 23_{10} and (-9_{10}) in binary using two's complement arithmetic .

$$\begin{array}{lrr} & 1\leftarrow\ 1\ 1\ 1\ \ 1\ 1\ 1 & \Leftarrow \text{carries} \\ \text{Discard} & 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1 & (23) \\ \text{carry.} & +\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1 & +(-9) \\ \hline & 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0 & 14_{10} \end{array}$$

With two's complement addition, the carry bit is _____.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.52. 26

Aside: Detecting Overflow

2

- Notice that the discarded carry in Example 8 did not cause an erroneous result.
- An overflow occurs if:
  - two positive numbers are added and the result is _____, or
  - two negative numbers are added and the result is _____.
- It is not possible to have overflow when using *two's complement notation* if a positive and a negative number are being added together.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.53. 27

2

- A simple rule for detecting an overflow condition using two's complement arithmetic (Assume 8-bit binary):

If the carry into the *sign bit* equals the carry out of the bit, _____ has occurred.

	1 ← 1 1 1 1 1 1 ⇐ carries	
Discard carry.	0 0 0 1 0 1 1 1	(23)
	+ 1 1 1 1 0 1 1 1	+ (–9)
	0 0 0 0 1 1 1 0	14_{10}

If the carry into the *sign bit* is different from the carry out of the sign bit, _____ (and thus an error) has occurred.

	0 ← 1 1 1 1 ⇐ carries	
Discard last carry.	0 1 1 1 1 1 1 0	(126)
	+ 0 0 0 0 1 0 0 0	+(8)
	1 0 0 0 0 1 1 0	(–122???)

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.53. 28

Module 2

Data Representation in Computer Systems

- Negation
- Addition
- Subtraction
- Multiplication
- Division

2.3 Fixed-Number Arithmetic

2

- This section examines common arithmetic functions on numbers in two's complement representation.

Negation

- In two's complement notation, the *negation* of an integer can be formed with the following rules:

1) Take the Boolean complement of each bit of the integer (including the sign bit) → Set each _____ and each _____.
2) Treating the result as an unsigned binary integer, add 1.

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.349. 30

2

- Those two-step process is referred to as the two's complement operation, or the taking of the two's complement of an integer.

+18	=	00010010 (twos complement)
bitwise complement	=	11101101
		+ 1
		11101110 = −18

−18	=	11101110 (twos complement)
bitwise complement	=	00010001
		+ 1
		00010010 = +18

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.349. 31

2

(a) Addition

- Digits are added bit by bit from right to left, with carries passed to the next digit to the left.

RECAP

Rules of Binary Addition

- 0 + 0 = 0 carry = 0
- 0 + 1 = 1 carry = 0
- 1 + 0 = 1 carry = 0
- 1 + 1 = 0 carry = 1

32

2

Example 9:

Addition of numbers in two's complement representation.

1001 = −7 +0101 = 5 1110 = −2 (a) (−7) + (+5)	1100 = −4 +0100 = 4 10000 = 0 (b) (−4) + (+4)
0011 = 3 +0100 = 4 0111 = 7 (c) (+3) + (+4)	1100 = −4 +1111 = −1 11011 = −5 (d) (−4) + (−1)

(e) and (f) show examples of overflow that can occur whether or not there is a carry.

0101 = 5 +0100 = 4 1001 = Overflow (e) (+5) + (+4)	1001 = −7 +1010 = −6 10011 = Overflow (f) (−7) + (−6)

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.351. 33

2

(b) Subtraction

- Subtraction is easily handled with the following rule:

- **SUBTRACTION RULE:** To subtract one number (_____) from another (_____), take the two's complement. (_____) of the *subtrahend* and add it to the *minuend*.

M (Minuend)
S (Subtrahend)

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.351. 34

M (Minuend)
S (Subtrahend)

2

Example 10:

Subtraction of numbers in two's complement representation $(M - S)$.

```
        0010 =  2          0101 =  5
       +1001 = −7         +1110 = −2
        1011 = −5         10011 =  3

(a) M =  2 = 0010  (b) M =  5 = 0101
    S =  7 = 0111      S =  2 = 0010
   −S =      1001     −S =      1110

        1011 = −5          0101 =  5
       +1110 = −2         +0010 =  2
       11001 = −7          0111 =  7

(c) M = −5 = 1011  (d) M =  5 = 0101
    S =  2 = 0010      S = −2 = 1110
   −S =      1110     −S =      0010

        0111 =  7          1010 = −6
       +0111 =  7         +1100 = −4
        1110 = Overflow   10110 = Overflow

(e) M =  7 = 0111  (f) M = −6 = 1010
    S = −7 = 1001      S =  4 = 0100
   −S =      0111     −S =      1100
```


2

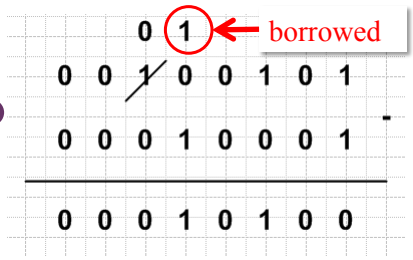
- Subtraction can be done directly.
- Rules of binary subtraction:

0 - 0 = 0
0 - 1 = 1, *and borrow 1 from the next more significant bit*
1 - 0 = 1
1 - 1 = 0

Example 11:

$37_{10} - 17_{10}$

(Assume 8 bit binary)



36

2 (c) Multiplication

- Multiplication is a complex operation, whether performed in hardware or software.
- The simpler problem of multiplying using unsigned integers, and the most common techniques for multiplication of numbers is two's complement representation.
- Multiplication of binary numbers must always use 0 and 1.

Rules of Binary Multiplication:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1
and no carry or borrow bit.

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.353. 37

2 (c) Multiplication Unsigned Integer

- Consist of operands called *multiplicand* and *multiplier*, and final result as *product.*

```
    Multiplicand
x   Multiplier
    ----------
    Product
    ----------
```

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.353. 38

2

Example 12:

Multiply the unsigned binary numbers of 1011_2 by 1101_2.

1011_2 → (11) *Multiplicand*
x 1101_2 → (13) *Multiplier*
1011
0000
1011
1011
10001111_2 → (143) *Product*

Partial products

If we ignore the sign bits, the length of multiplication of an n-bit _____ and an m-bit _____ is a _____ that is $(n+m)$ bit long.

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.353. 39

2

(c) Multiplication

Signed Integer: Two's Complement

- We have seen that addition and subtraction can be performed on numbers in two's complement notation by treating them as unsigned integers.
- **Example**:

1001
+ 0011
1100

If these numbers are considered to be unsigned integers, then we are adding 9 (1001_2) plus 3 (0011_2) to get ____ (1100_2)

As two's complement integers, we are adding –7 (1001_2) to 3 (0011_2) to get _____ (1100_2).

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.355. 40

2

- Unfortunately, this simple scheme will not work for multiplication.

Example 13:

To see this, consider again **Example 12**.

Multiply the two's complement binary numbers of 1011_2 by 1101_2.

1011_2 → (–5) *Multiplicand*
x 1101_2 → (–3) *Multiplier*
1011
0000
1011
1011
10001111_2 → (–113) *Product*

Partial products

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.356. 41

2

- This example demonstrates that straightforward multiplication will not work if both the multiplicand and/or multiplier are negative.

Regular multiplication clearly yields incorrect result !

- **Solution** : _____ algorithm.

  This algorithm has the benefit of speeding up the multiplication process, relative to a more straightforward approach.

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.353. 42

A Multiplication Algorithm and Hardware

2

- Multiplication must cope with overflow because we frequently want a 32-bit *product* as the result of multiplying two 32-bit numbers.
- In the next slides, assume that we are multiplying only _____ number (unsigned) with the 1st version of highly optimized multiplication hardware.

43

2

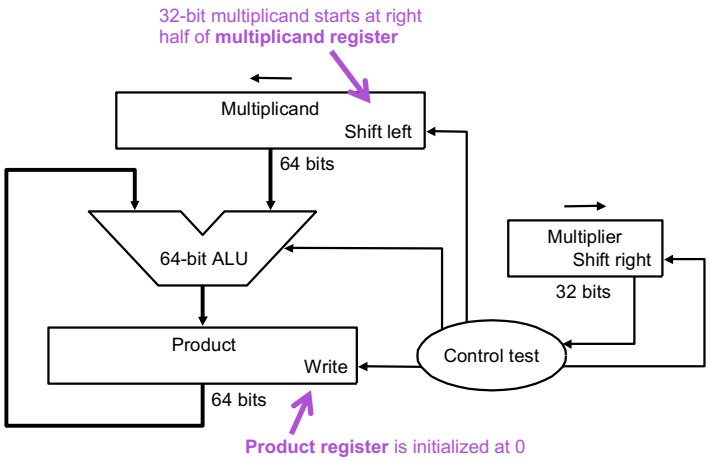


Figure: First version of Multiplication Hardware.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.184. 44

Example 14:

Using 4-bit numbers, multiply of 2_{10} x 3_{10} .

2 x 3 = ?

0010_2 x 0011_2

Steps:
1 – Test multiplier (0 or 1)
If 1 then 1a: P = P + MC
If 0 then no operation
2 – Shift MC left
3 – Shift MP right
All bits done?
If still <max bit, repeat
If = max bit, stop

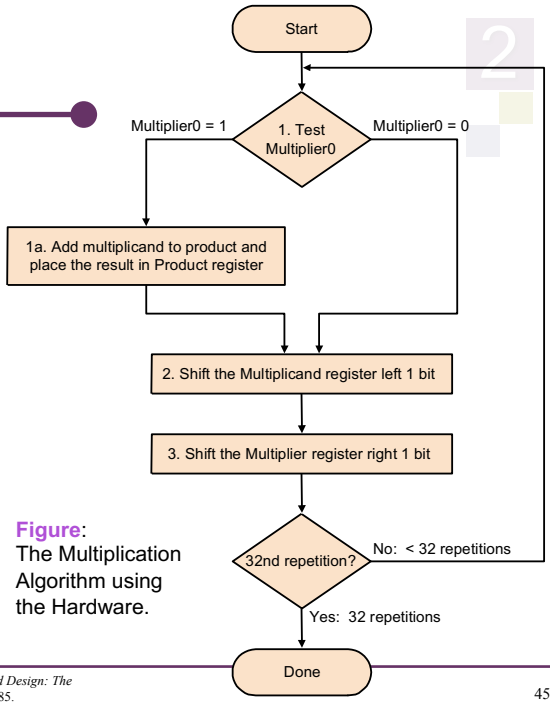


Figure: The Multiplication Algorithm using the Hardware.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.185.

45

2_{10} x 3_{10} = _____$_{10}$

Iteration	Step	Multiplier (MP)	Multiplicand (MC)	Product (P)
0	Initial value	0011	0000 0010	0000 0000
1	1a: P = P + MC	001**1**		0000 0010
	2: Shift MC left		0000 0100	
	3: Shift MP right	000**1**		
2	1a: P = P + MC	000**1**		0000 0110
	2: Shift MC left		0000 1000	
	3: Shift MP right	0000		
3	1: No Operation	0000		
	2: Shift MC left		0001 0000	
	3: Shift MP right	0000		
4	1: No Operation	0000		
	2: Shift MC left		0010 0000	
	3: Shift MP right	0000		

Answer:
0000 0110
= 6_{10}

Aside:

2

- The *multiplier* (MP) must always in positive number.
- Do an *additive inverse* to the *multiplicand* (MC) and the MP.

$$Multiplicand \times (-Multiplier) = (-Multicand) \times Multiplier$$
$$MC \times (-MP) = (-MC) \times MP$$

- **Examples**:

$$7 \times (-5) = (-7) \times 5$$
$$(-7) \times (-5) = 7 \times 5$$

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.53. 47

2

Example 15:

Using a 4-bit binary arithmetic, multiply 2_{10} with (-3_{10}) using the 1st version of highly optimized multiplication hardware.

Solution:

→ $2 \times (-3)$

- Do an _____ to the multiplicand (MC) and the MP:

$$(-2) \times 3$$

- Perform the multiplication as usual.

48

$(-2_{10}) \times 3_{10} =$ _____$_{10}$

Iteration	Step	Multiplier (MP)	Multiplicand (MC)	Product (P)
0	Initial value	0011	1111 1110	0000 0000
1				
	2: Shift MC left			
	3: Shift MP right			
2				
	2: Shift MC left			
	3: Shift MP right			
3				
	2: Shift MC left			
	3: Shift MP right			
4				
	2: Shift MC left			
	3: Shift MP right			

2

(d) Division

- More complex than multiplication but is based on the same general principles.
- An operation that is even less frequent and even more quickly.
- It even offers the opportunity to perform a mathematically invalid operations in dividing by 0.
- Two operands called _____ and *divisor*, the result as _____ with secondary result called *remainder*.

```
          Quotient
Divisor  ⟌Dividend   .
          ...
             ...    .
          Remainder
```

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.360. 50

2

- Another way to express the relationship between the components:

$$Dividend = Quotient \times Divisor + Remainder$$

where the _____ is smaller than the _____.

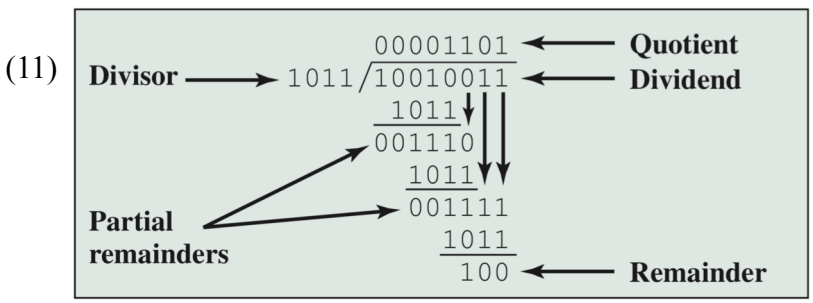
- Infrequently, programs use the divide instruction just to get the *remainder*, ignoring the *quotient*.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.193. 51

2

(d) Division
Unsigned Integer

- The following figure shows an example of the long division of unsigned binary integers of 147_{10} divided by 11_{10}.



William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.348. 52

2

(d) Division

Signed Integer: Two's Complement

- Make both *dividend* and *divisor* positive and perform division.
- Make the sign of the *remainder* match to the *dividend*, no matter what the signs of the *divisor* and *quotient*.
- The rules:

$$+7 \div +2: Quotient = +3, Remainder = +1$$
$$+7 \div -2: Quotient = -3, Remainder = +1$$
$$-7 \div +2: Quotient = -3, Remainder = -1$$
$$-7 \div -2: Quotient = +3, Remainder = -1$$

- *Negate* the *quotient* if *dividend* and *divisor* were of opposite signs.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.193. 53

2

A Division Algorithm and Hardware

- Binary division is restricted to 0 or 1, thereby simplifying binary division.
- In the next slides, assume that both the *dividend* and *divisor* are positive number; Hence the *quotient* and *remainder* are non-negative.
- Since iteration of the algorithm needs to move the *divisor* to the right one digit, we start the *divisor* placed in the left half of the 64-bit Divisor Register.

54

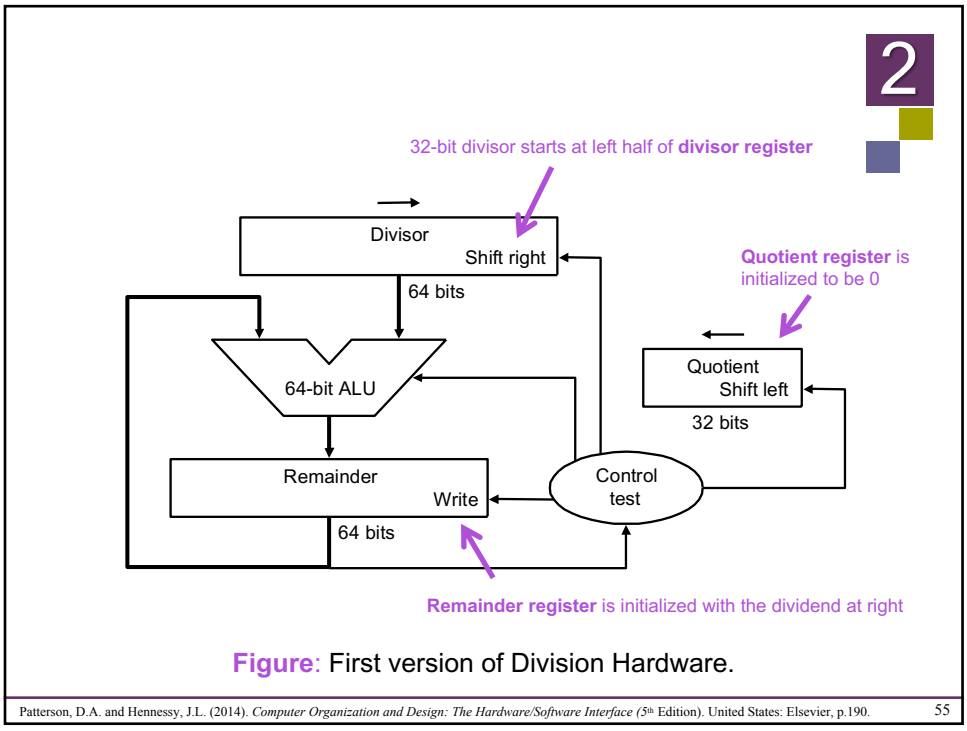


Figure: First version of Division Hardware.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.190. 55

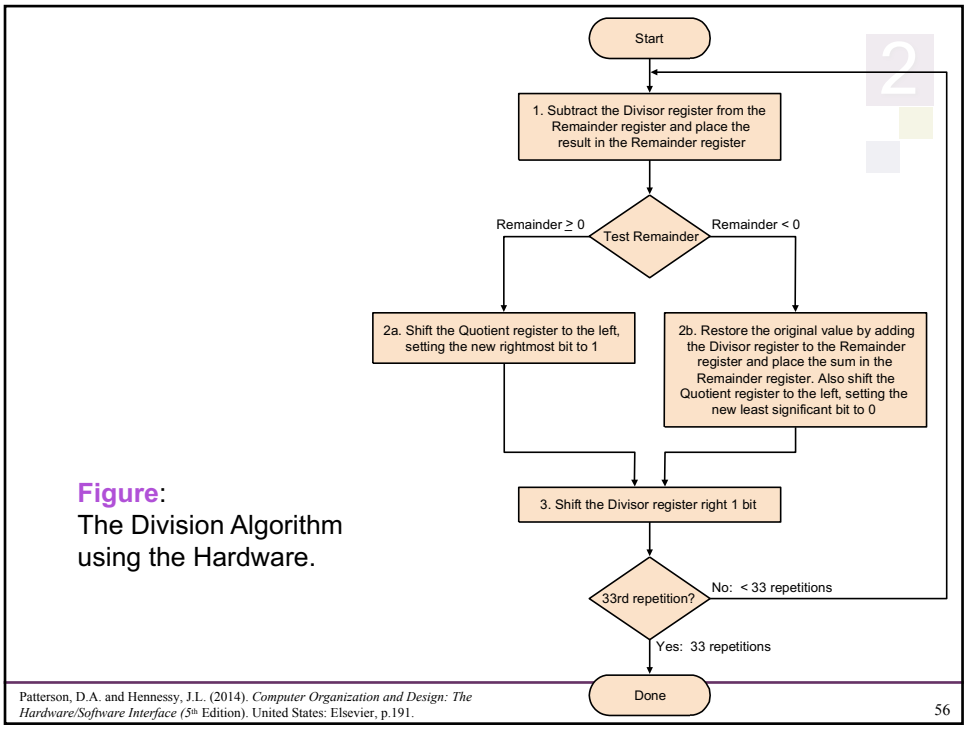


Figure:
The Division Algorithm using the Hardware.

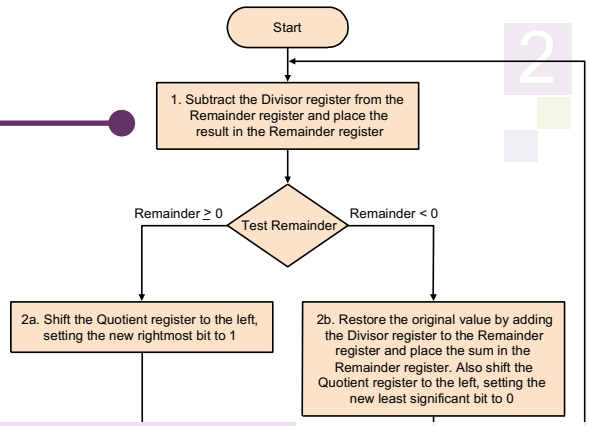
Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.191. 56

Example 16:

Using 4-bit numbers, divide 7_{10} by 2_{10}.

7 / 2 = ?

0111_2 / 0010_2



Steps:
1 – Remainder (R) = R – D
2 – test new R
2a - If >=0 then Shift left Q (add 1 at LSB)
2b - If <0 then R = D + R, Shift left Q (add 0 at LSB)
3 – shift D right
All bits done?
If still <(max bit + 1), repeat
If = (max bit + 1), stop

57

7 / 2 = ____

Iteration	Steps	Quotient (Q)	Divisor (D)	Remainder (R)
0	Initial value	0000	0010 0000	0000 0111
1	1 : R = R – D			1110 0111
	2b: R < 0; R = D+R			0000 0111
	Q : Shift Left (+0)	0000		
	3 : D = Shift right		0001 0000	
2	1 : R = R – D			1111 0111
	2b: R < 0; R = D+R			0000 0111
	Q : Shift Left (+0)	0000		
	3 : D = Shift right		0000 1000	
3	1 : R = R – D			1111 1111
	2b: R < 0; R = D+R			0000 0111
	Q : Shift Left (+0)	0000		
	3 : D = Shift right		0000 0100	

58

2

Try to complete the table for the remaining iterations:
R = 0000 0111_2; Q = 0000_2; D = 0000 0100_2

Iteration	Steps	Quotient (Q)	Divisor (D)	Remainder (R)
4	1 : R = R – D			
	3 : D = Shift right			
5	1 : R = R – D			
	3 : D = Shift right			

59

2

60

Module 2

Data Representation in Computer Systems

- Representation
- Components
- Normalized & Unnormalized
- Format: A Simple Model
- The IEEE-754 Floating-Point Standard

2.4 Floating-Point Representation

2

- The signed magnitude, one's complement, and two's complement representation that we have just presented deal with integer values only.
- Without modification, these formats are not useful in scientific or business applications that deal with real number values.
- Floating-point representation solves this problem.
- Known as _____ in mathematics.

62

2 Representation

- Floating-point numbers allow an arbitrary number of decimal places to the right of the decimal point.
  - **Examples**:
  -

$$5 \times 25 = 125$$
$$0.5 \times 0.25 = 0.125$$

→ often expressed in scientific notation as:

$$125 = 1.25 \times 10^{2}$$
$$0.125 = 1.25 \times 10^{-1}$$

63

Sign
Significand/Fraction/Mantissa
Base/Radix
Exponent

2 Components

(Decimal) $+743.059 \times 10^{-63}$

(Binary) -101.001×2^{011}

Fraction and *exponent* can be +ve or –ve

64

2 Normalized & Unnormalized

- In generalized normalization (like in mathematics), a floating point number is said to be _____ if the number after the radix point is a non-zero value.
- _____ floating number is when the number after the radix point is '0'.
- **Examples**:

1.03×10^{-9} $(Unnormalized)$

0.10×10^{8} $(Normalized)$

45.07×10^{-10}

0.1234×10^{16}

65

2 Normalization Process

- *Normalization* is the process of deleting the zeroes until a non-zero value is detected.
- **Examples**:

$$0.00743 \times 10^{4} = 0.743 \times 10^{4-2} = 0.743 \times 10^{2}$$

$$45.09 \times 10^{4} = 0.4509 \times 10^{4+2} = 0.4509 \times 10^{6}$$

A rule of thumb:
- moving the radix point to the right → _____ exponent
- moving the radix point to the left → _____ exponent

66

2

Examples:

- Decimal:

$$743.09 \times 10^{61} = 0.74309 \times 10^{61+3}$$
$$= 0.74309 \times 10^{64}$$

- Binary:

$$10.0111_2 \times 2^{-110011} = 0.100111_2 \times 2^{(-110011)+010}$$
$$= 0.100111_2 \times 2^{-110001}$$

$$0.00011011_2 \times 2^{9} = 0.11011_2 \times 2^{9-3}$$
$$= 0.100111_2 \times 2^{6}$$

67

2

Format : A Simple Model

- In digital computers, floating-point numbers consist of three parts: a *sign bit*, an *exponent* part (representing the exponent on a power of 2), and a fractional part called a _____.
- The general form of *normalized* floating-point is:

$$\pm 0.Fraction \times Base^{\pm exponent}$$

- In binary form:

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.56. 68

2

Figure: Example of 14-bit model for floating-point format.

- The next few examples and exercises will be used this simple model before the discussion on the IEEE-754 floating-point standard.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.56. 69

Example 17:

Store the decimal number 17 in this model.

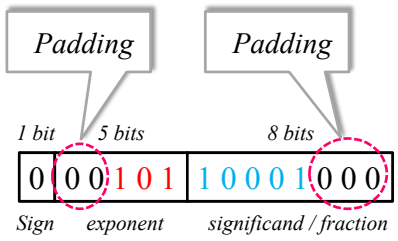
Solution:

(Unnormalized Binary):

$17 = 10001.0 \times 2^0$

(Normalized Binary):

$= 1000.1 \times 2^1$
$= 100.01 \times 2^2$
$= 10.001 \times 2^3$
$= 1.0001 \times 2^4$
$= 0.10001 \times 2^5$



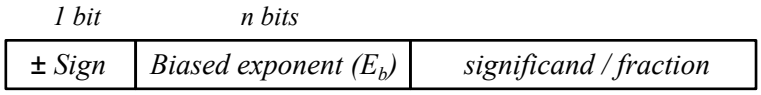
Rule of thumb:
- the *exponent* is always padded to the left (←).
- the *fraction* is always padded to the right (→).

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.56-57. 70

Biased Exponent

This is used unless the IEEE standard is mentioned – then is a different calculation

- One obvious problem with this model is that negative exponents cannot be represented.
- **Solution**: _____



Where,
E_b = *biased exponent*
n = *bits of exponent format (i.e. the word format)*

Biased value, $b = 2^{n-1}$
Normalized exponent, $e' = E_b - b$
Biased exponent, $E_b = e' + b$

71

2

- The _____ → a number near the middle of the range of possible values that we select to represent zero.
- Typically, the bias value (b) is equals (2^{n-1}), where n is the number of bits in the binary exponent.
  - *positive* value : Any number larger than (2^{n-1}) in the exponent field → bias value + exponent
  - *negative* value : Any number less than (2^{n-1}) in the exponent field → bias value - exponent

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.57.
William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.328.

72

2

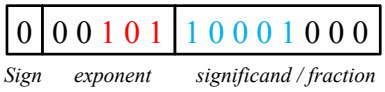
- The steps of conversion to floating-point:
  - Change to binary (if given decimal number).
  - [ ] the number.
  - Change the number to *biased exponent*.
  - Form the floating-point format (3 fields) .

73

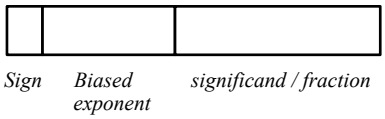
2

Example 18:

Returning to *Example 17* that storing: $17 = 0.10001 \times 2^5$



- The *bias* value, $b = 2^{5-1} = 2^4 = 16$
- The *biased exponent*, E_b is now = $e' + b = 5 + 16 = 21$



74

Example 19:

Store the decimal number 0.25 in the floating-point representation.

Show your working.

Solution:

(Convert into binary using repetitive multiplication):

$0.25 \times 2 = 0.5$

$0.5 \times 2 = 1.0$

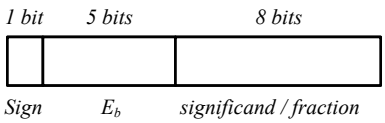
$0.25_{10} = 0.01_2$

(Normalized Binary):

$0.25_{10} = 0.01 \times 2^0$

$= 0.1 \times 2^{-1}$

- The *bias* value, $b = 2^{5-1} = 16$
- The *biased exponent*, E_b is now 16 + (–1) = 15

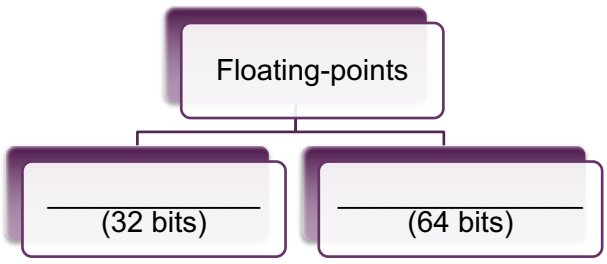


75

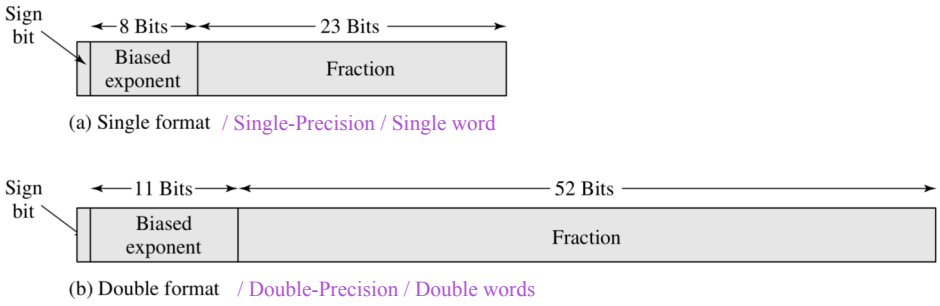
2 The IEEE-754 Floating-point Standard

(IEEE) Institute of Electrical and Electronic Engineers

- The floating-point model (non-standard) described before is for simplicity and conceptual understanding.
- This standard is officially known as IEEE-754 (1985):



William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.331. 76



(a) Single format / Single-Precision / Single word

(b) Double format / Double-Precision / Double words

Figure: IEEE-754 floating-point format.

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.331. 77

MIPS (Millions of Instructions Per Second)

Overview

- *Floating-point* → Computer arithmetic that represents numbers in which the binary point is not fixed:

$$1.xxxxxxxx_2 \times 2^{yyyy}$$

- A standard scientific notation for reals in normalized form.
- General form: $(-1)^S \times Fraction \times 2^{e'}$

where $S = Sign$, e' = Normalized exponent.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.197. 78

MIPS (Millions of Instructions Per Second)

2

(a) Single Precision

- The representation of a MIPS 32-bit floating-point number:

- The sizes of *exponent* and *fraction* give MIPS computer arithmetic an extraordinary range:
  - Smallest fraction: $2.0_{10} \times 10^{-38}$
  - Largest number: $2.0_{10} \times 10^{38}$
- This is called _____ = 1 word.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.198. 79

2

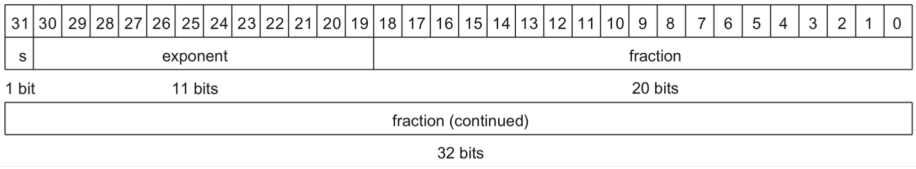
- Unfortunately, single precision floating-point still possible for numbers to be too large:
  - _____ → the *exponent* is too large.
  - _____ → the negative *exponent* is too large.
- **Solution**:

  One way to reduce these problems, need another format that has larger exponent → _____ floating-point.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.198. 80

(b) Double Precision

- The representation of a double precision floating-point number takes two MIPS words → 64 bits.



  - Smallest fraction: $2.0_{10} \times 10^{-308}$
  - Largest number: $2.0_{10} \times 10^{308}$

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.198. 81

The IEEE-754 Floating-point Standard Representation

- This standard has greatly improved both the ease of porting floating-point programs and the quality of computer arithmetic.
- To pack even more bits into the *significand*, IEEE 754 makes the leading 1-bit of normalized binary numbers implicit.

- *Significant* : 1 + fraction
  - *Single precision* : 24 bits
  - *Double precision* : 53 bits

0 has no leading 1, it is given the reserved exponent value 0.

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.198. 82

2

- The representation of normalized floating-point in IEEE standard:

$$(-1)^S \times (1 + Fraction) \times 2^{e'}$$

Sign (S): plus (0) or minus (1)
Normalized exponent (e')

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.199. 83

2

The IEEE-754 Floating-point Standard
Normalized & Unnormalized

- In IEEE standard normalization (used in computers), a floating point number is said to be _____ if there is only a single non-zero before the radix point.
- **Examples**:

1.03×10^{-9}	$(Normalized)$
0.10×10^{8}	$(Unnormalized)$
1.011×2^{-011}	
1.234×10^{16}	

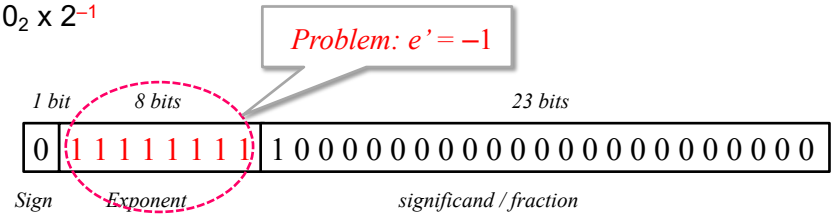
84

Biased Notation

- Placing the exponent before the fraction simplifies sorting of floating-point numbers using integer comparison instructions.
- However, using 2's complement in the exponent field makes a negative exponent look like a big number.

- **Example**: IEEE-754 Single Precision (32 bits)

  $1.0_2 \times 2^{-1}$



85

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

Sign (S): plus (0) or minus (1)
Biased Exponent (E_B)
Bit of Exponent (n)

Bias values, $B = (2^{n-1}) - 1$:

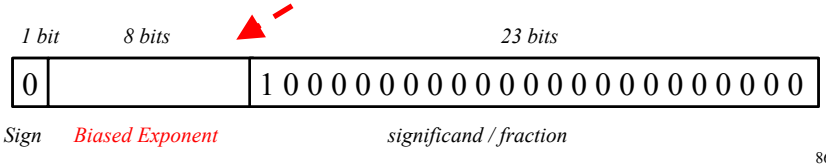
→ In *single precision* is _____

→ In *double precision* is _____

- **Example**:

  $1.0_2 \times 2^{-1}$

$E_B = e'$ + Bias
=
=



86

2

The IEEE-754 Floating-point Standard

Conversion of Decimal to Binary Floating-point

- To convert a decimal number to single or double precision floating point:
  - Step 1: Normalized.
  - Step 2: Determine sign bit.
  - Step 3: Determine *biased exponent*.
  - Step 4: Determine significand (fraction)

87

2

Example 20:

Convert 10.4_{10} to single precision floating-point.

- **Step 1**: Normalized from the bit value.

$10_{10} \rightarrow 1010_2$

0.4 x 2 = 0.8
0.8 x 2 = 1.6
0.6 x 2 = 1.2
0.2 x 2 = 0.4
.....

(Repetitive Multiplication)

10.4_{10} =
=
=

- **Step 2**: Determine sign bit (S). → S = 0

88

2

- **Step 3**: Determine the biased exponent, E_B →

$E_B = e'$ + bias
=
=

- **Step 4**: Determine fraction.
  - ❑ Drop the leading 1 of the fraction;

$1.0100110_2 \times 2^{10000010}$ → 0100110

  - ❑ Expand (padding) to 23 bits;

01001100000000000000000

  - ❑ Complete the representation;

89

2

The IEEE-754 Floating-point Standard

Conversion of Binary Floating-point to Decimal

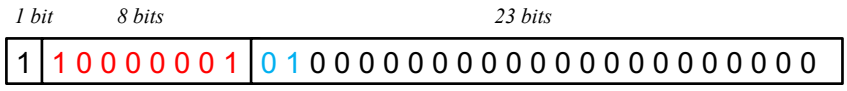
- To convert a single or double precision floating point to decimal number:
  - **Step 1**: Extract value of sign.
  - **Step 2:** Extract value of & bias value.
  - **Step 3**: Extract value of fraction.
  - **Step 4**: Apply the basic equation.

90

2

Example 21:

What is the decimal number represented by this single precision float?



Solution:

- **Step 1**: Extract value of sign. → S = _____
- **Step 2**: Extract value of biased exponent, E_B → _____ = ____10

  Bias value, B → $(2^{n-1} - 1) = 2^7 - 1 = 128 - 1 = 127_{10}$

$S = 1$
$Fraction = 0.25$
$E_B = 129$
$B = 127$

- **Step 3**: Extract value of fraction. → 1 x 2^{-2} $= \frac{1}{2^2} = \frac{1}{4} = 0.25_{10}$
- **Step 4**: Apply the basic equation.

$$(-1)^S \times (1 + Fraction) \times 2^{e'}$$

= $(-1)^1$ x (1 + 0.25) x $2^{129-127}$

= (– 1) x (1.25) x 2^2

= (– 1.25) x 4

= -5.0_{10}

Module 2

Data Representation in Computer Systems

- Overview
- Addition
- Multiplication

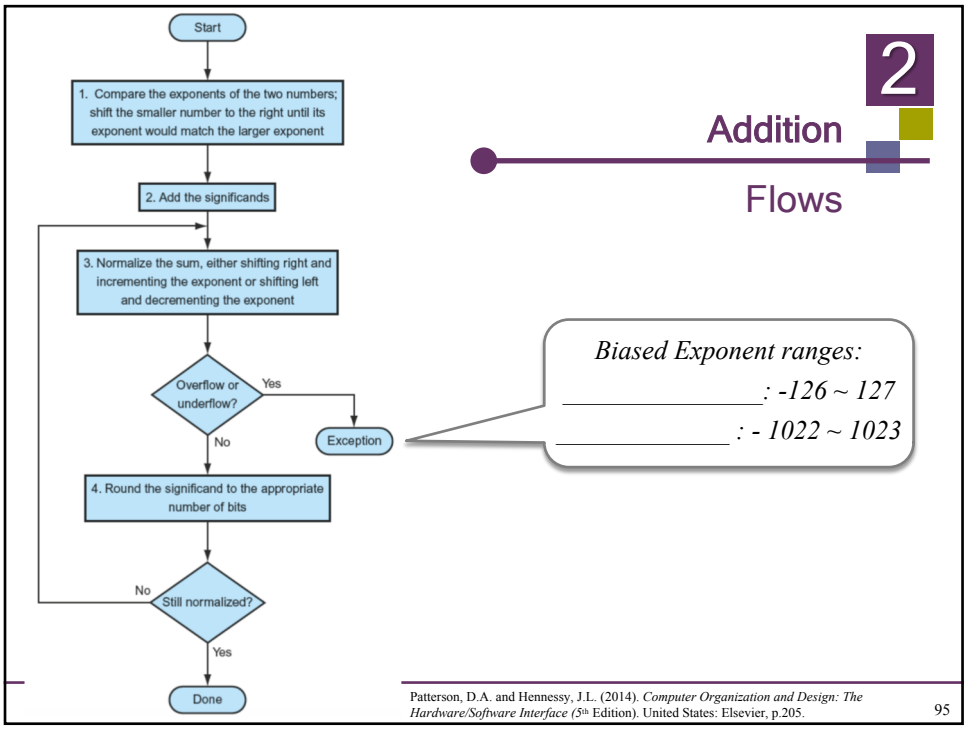
2.5 Floating-Point Arithmetic

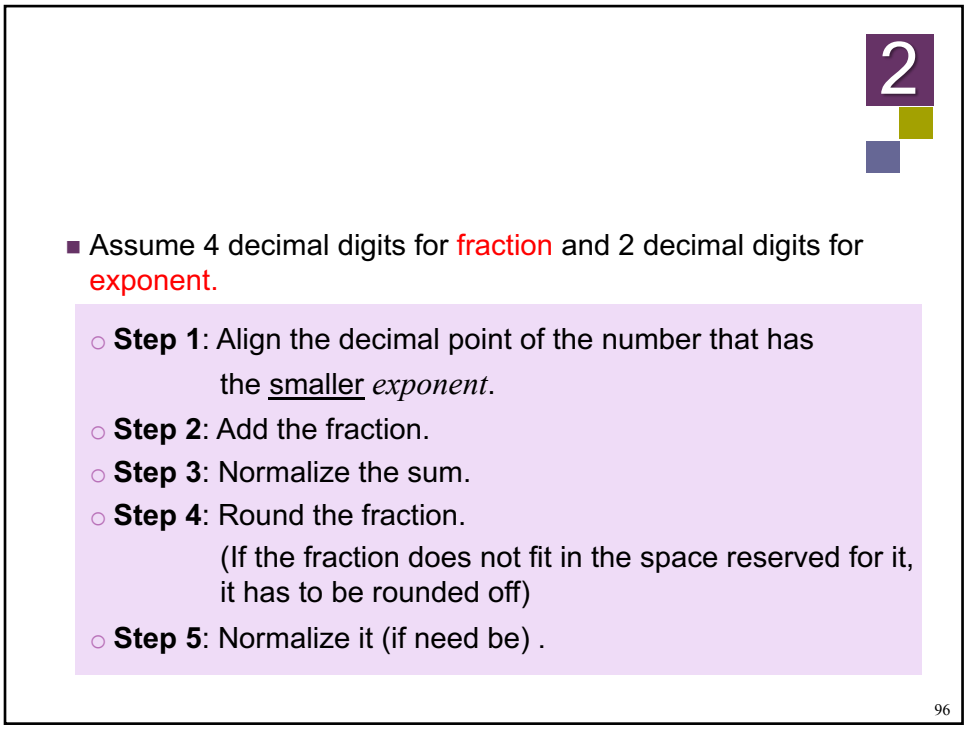
2

Overview

- For addition and subtraction, it is necessary to ensure that both operands have the same exponent value.
- This may require shifting the radix point on one of the operands to achieve alignment.
- Multiplication and division are more straightforward.

William Stallings (2013). *Computer Organization and Architecture: Designing for Performance* (9th Edition). United States: Pearson Education Limited, p.334. 94


Start
1. Compare the exponents of the two numbers; shift the smaller number to the right until its exponent would match the larger exponent
2. Add the significands
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow or underflow?
Yes
No
Exception
4. Round the significand to the appropriate number of bits
No
Still normalized?
Yes
Done
2
Addition
Flows
Biased Exponent ranges:
_____: -126 ~ 127
_____ : - 1022 ~ 1023
Patterson, D.A. and Hennessy, J.L. (2014). Computer Organization and Design: The Hardware/Software Interface (5th Edition). United States: Elsevier, p.205.
95


2
Assume 4 decimal digits for fraction and 2 decimal digits for exponent.
Step 1: Align the decimal point of the number that has the smaller exponent.
Step 2: Add the fraction.
Step 3: Normalize the sum.
Step 4: Round the fraction.
(If the fraction does not fit in the space reserved for it, it has to be rounded off)
Step 5: Normalize it (if need be) .
96

Example 22:

Add these two decimal floating-point numbers. Assume that we can store only four decimal digits of the significand and two decimal digits of the exponent.

$(9.999_{10} \times 10^{1}) + (1.610_{10} \times 10^{-1}) = \_\_\_\_\_{}_{10}$

Solution:

- **Step 1**: Align the decimal point of the number that has the smaller exponent.

$1.610_{10} \times 10^{-1}$

$= 1.610_{10} \times 10^{-1} \times 10^{2}$

$= 0.0161_{10} \times 10^{1}$

- **Step 2**: Add the fraction.

$$\begin{array}{r} 9.9990 \times 10^{1} \\ +\ 0.0161 \times 10^{1} \\ \hline 10.0151 \times 10^{1} \end{array}$$

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.203. 97

2

- **Step 3**: Normalize the sum.

$10.0151 \times 10^{1} \rightarrow 1.00151 \times 10^{2}$

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

$1.00151 \times 10^{2} \rightarrow 1.0015 \times 10^{2}$

- **Step 5**: Normalize it (if need be).

No need as its normalized

Answer = 1.0015×10^{2}

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.203. 98

2

Example 23:

Add these two binary floating-point numbers.

$0.5_{10} + (-0.4375_{10}) =$ _____ $_2$

Solution: (Convert to binary)

0.5 x 2 = 1.0

$(0.1_2 \times 2^0) \times 2^{-1}$

$(-0.0111_2 \times 2^0) \times 2^{-2}$

$(1.0_2 \times 2^{-1}) + (-1.11_2 \times 2^{-2})$

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.204. 99

- **Step 1**: Align the decimal point of the number that has the smaller exponent.

  $-1.11_2 \times 2^{-2}$

  $= -1.11_2 \times 2^{-2} \times 2^1$

  $= -0.111_2 \times 2^{-1}$

- **Step 2**: Add the fraction.

  ~~1~~.~~000~~ x 2^{-1}

  + – 0.111 x 2^{-1}

  0.001 x 2^{-1}

- **Step 3**: Normalize the sum.

  0.001 x 2^{-1} x 2^{-3} → 1.0 x 2^{-4}

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

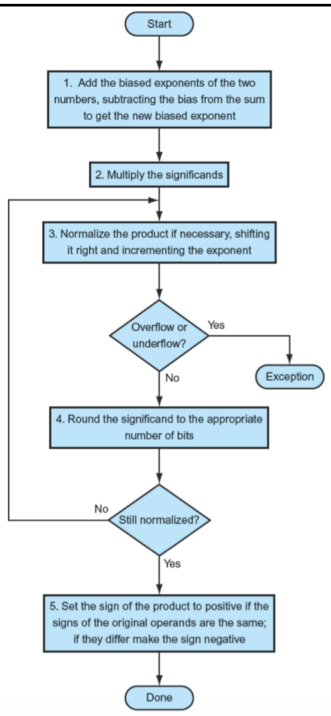
  1.0 x 2^{-4} → 1.0000 x 2^{-4}

- **Step 5**: Normalize it (if need be).

  No need as its normalized

Answer = _____

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface* (5th Edition). United States: Elsevier, p.204. 100



2

Multiplication

Flows

Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.209.

101

2

- Assume 4 decimal digits for fraction and 2 decimal digits for exponent.
  - **Step 1**: Add the *exponent* of the 2 numbers.
  - **Step 2**: Multiply the fraction.
  - **Step 3**: Normalize the product.
  - **Step 4**: Round the fraction.
    (If the fraction does not fit in the space reserved for it, it has to be rounded off)
  - **Step 5**: Normalize it (if need be) .
  - **Step 6**: Set the sign of the product.

102

2

Example 24:

Multiply these two decimal floating-point numbers.

Assume 4 decimal digits for *significand* and 2 decimal digits for *exponent*.

$$(1.110 \times 10^{10}) \times (9.200 \times 10^{-5}) = \_\_\_\_\_{}_{10}$$

Solution:

- **Step 1**: Add the exponent of the 2 numbers.

  10 + (–5) = 5

  If bias considered →

  5 + 127 = 132

- **Step 2**: Multiply the fraction.

```
    9.200
 x  1.110
    0 000
   92 00
  920 0
 9200
10212000
```

→ 10.212000
= 10.2120 x 10^5

103

2

- **Step 3**: Normalize the product.

  10.2120 x 10^5 x 10^1 → 1.02120 x 10^6

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

  1.02120 x 10^6 → 1.0212 x 10^6

- **Step 5**: Normalize it (if need be).

  No need as its normalized

- **Step 6**: Set the sign of the product.

  +1.0212 x 10^6

104

2

Example 25:

Multiply these two binary floating-point numbers.
Assume 4 binary digits for significand and 2 binary digits for exponent.

$$(1.000 \times 2^{-1}) \times (-1.110 \times 2^{-2}) \quad = \_\_\_\_\_ _2$$

Solution:

- **Step 1**: Add the exponent of the 2 numbers.

  (–1) + (–2) = –3

  If bias considered →

  (–3) + 127 = 124

- **Step 2**: Multiply the fraction.

```
    1.110
 x  1.000
    0 000
   00 00
  000 0
 1110
1110000
```

→ 1.110000
= 1.110000 x 2^{-3}

105

2

- **Step 3**: Normalize the product.

  Already normalized

- **Step 4:** Round the fraction (to 4 decimal digits for fraction).

  1.110000 x 2^{-3} → 1.1100 x 2^{-3}

- **Step 5**: Normalize it (if need be).

  No need as its normalized

- **Step 6**: Set the sign of the product.

  –1.1100 x 2^{-3}

2.6 Summary

2

- This module presented the essentials of data representation and numerical operations in digital computers.
- Student should master the techniques described for base conversion and memorize the smaller hexadecimal and binary numbers.
- This knowledge will be beneficial to student throughout remainder of this subject.
- Knowledge of hexadecimal coding will be useful if you are ever required to read a core (memory) dump after a system crash or if do any serious work in the field of data communications.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.83. 107