The slide features a blue background with a grid pattern and various computer-related icons. The text is centered and includes the course name and date.

**Lecture 1:
Introduction and Number Representations**

ECE 645—Computer Arithmetic
1/22/08

ECE 645 – Computer Arithmetic

Lecture Roadmap

- Syllabus and Course Objectives
- ECE 645 CAD Tools
- Computer Arithmetic: Introduction
- Fixed-Point Number System Representations
 - Fixed-Radix Unsigned Representations
 - Fixed-Radix Signed Representations
 - Signed-Magnitude
 - Biased
 - Digit Complement (One's Complement)
 - Radix Complement (Two's Complement)

Required Reading

- B. Parhami, "Computer Arithmetic: Algorithms and Hardware Design"
 - Chapter 1, Numbers and Arithmetic (entire chapter)
 - Chapter 2, Representing Signed Numbers (entire chapter)
- Note errata at:
 - http://www.ece.ucsb.edu/~parhami/text_comp_arit.htm#errors



The graphic features a central image of a 'CoolRunner-II' integrated circuit. Surrounding it are several circular icons: a laptop, a mobile phone, a keyboard, and a mouse. The background is a blue-tinted circuit board with the words 'High Performance', 'CoolClock', 'Low Power', and 'DataGate' written in a stylized font. The text 'Syllabus and Course Objectives' is prominently displayed in the center.

Syllabus and Course Objectives

ECE 645 – Computer Arithmetic

About the Instructor

- Dr. David Hwang
 - PhD in Electrical Engineering, UCLA 2005
 - Thesis: System Architectures and VLSI Implementations of Secure Embedded Systems
 - Worked in industry designing VLSI signal processing algorithms and circuits
 - Research Interests
 - Secure embedded systems
 - Cryptographic hardware and circuits
 - VLSI digital signal processing
 - VLSI systems and circuits

5

Course Objectives

- At the end of this course you should be able to:
 - Understand mathematical and gate-level algorithms for computer addition, multiplication, and division
 - Understand tradeoffs involved with different arithmetic architectures between performance, area, latency, etc.
 - Understand sources of error in computer arithmetic and error analysis
 - Be comfortable with different number systems, and have familiarity with Galois field and finite field arithmetic for future study
 - Synthesize and implement computer arithmetic blocks on FPGAs
- This knowledge will come about through homework, exams, and projects

6



FPGA CAD Tools

- This class assumes proficiency with the FPGA CAD tools from ECE 545
 - As a refresher, go to last semester's ECE 545 and run through the hands-on sessions
- You are expected to be proficient with:
 - Synthesizable VHDL coding
 - Advanced VHDL testbenches, including file input/output
 - Xilinx FPGA synthesis and post-synthesis simulation
 - Xilinx FPGA place-and-route and post-place and route simulation
 - Reading and interpreting all synthesis and implementation reports

ECE 645 CAD Tool Flows

Environment	Simulation	Synthesis	Implementation
Aldec Active-HDL 7.2 SP2	Aldec Active-HDL 7.2 SP2	Synplicity Synplify Pro 8.6.2	Xilinx ISE Foundation 9.1 SP3
"	"	Xilinx XST 9.1 SP3	"
Xilinx ISE Foundation 9.1 SP3	Mentor Graphics Modelsim SE 6.3a	Synplicity Synplify Pro 8.6.2	Xilinx ISE Foundation 9.1 SP3
"	"	Xilinx XST 9.1 SP3	"

- The above four design flows are all installed on the lab computers in ST2 203 and ST2 265
- The two design flows using XST can also be emulated on your laptop or home computer using the techniques shown on the web site:

9

**Computer Arithmetic:
Introduction**

ECE 645 - Computer Arithmetic

Computer Arithmetic Advances

Proceedings of conferences

ARITH - *International Symposium on Computer Arithmetic*

ASIL - *Asilomar Conference on Signals, Systems, and Computers*

ICCD - *International Conference on Computer Design*

CHES - *Workshop on Cryptographic Hardware and Embedded Systems*

Journals and periodicals

IEEE Transactions on Computers,

in particular special issues on computer arithmetic:

8/70, 6/73, 7/77, 4/83, 8/90, 8/92, 8/94, 7/00, 3/05.

IEEE Transactions on Circuits and Systems

IEEE Transactions on Very Large Scale Integration

IEE Proceedings: Computer and Digital Techniques

Journal of VLSI Signal Processing

11

What is Computer Arithmetic?

Hardware (our focus in this class)

Design of efficient digital circuits for primitive and other arithmetic operations such as $+$, $-$, \times , \div , $\sqrt{\quad}$, \log , \sin , \cos

Issues: Algorithms
Error analysis
Speed/cost trade-offs
Hardware implementation
Testing, verification

General-purpose

Flexible data paths
Fast primitive operations like $+$, $-$, \times , \div , $\sqrt{\quad}$
Benchmarking

Special-purpose

Tailored to applications like:
Digital filtering
Image processing
Radar tracking

Software

Numerical methods for solving systems of linear equations, partial differential equations, etc.

Issues: Algorithms
Error analysis
Computational complexity
Programming
Testing, verification

- From Parhami's slides
-

12

Approximations and Error

- In digital arithmetic one has to come to grips with approximation and questions like:
 - When is approximation good enough
 - What margin of error is acceptable
- Be aware of the applications you are designing the arithmetic circuit or program for
- Analyze the implications of your approximation

13

Calculators

$$u = \underbrace{\sqrt{\sqrt{\sqrt{\dots\sqrt{2}}}}}_{10 \text{ times}} = 1.000\ 677\ 131$$

$$x = \underbrace{(((u^2)^2)\dots)^2}_{10 \text{ times}} = 1.999\ 999\ 963$$

$$x' = u^{1024} = 1.999\ 999\ 973$$

$$v = 2^{1/1024} = 1.000\ 677\ 131$$

$$y = \underbrace{(((v^2)^2)\dots)^2}_{10 \text{ times}} = 1.999\ 999\ 983$$

$$y' = v^{1024} = 1.999\ 999\ 994$$

Hidden digits in the internal representation of numbers
 Different algorithms give slightly different results
 Very good accuracy

14

Consequences of Bad Approximations

Example: Failure of Patriot Missile (1991 Feb. 25)

Source <http://www.math.psu.edu/dna/455.f96/disasters.html>

American Patriot Missile battery in Dharan, Saudi Arabia, failed to intercept incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28

Cause, per GAO/IMTEC-92-26 report: "software problem" (inaccurate calculation of the time since boot)

Specifics of the problem: time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to get the time in seconds. Internal registers were 24 bits wide 1/10 = 0.0001 1001 1001 1001 100 (chopped to 24 b)

Error $\cong 0.1100\ 1100 \times 2^{-23} \cong 9.5 \times 10^{-8}$

Error in 100-hr operation period

$\cong 9.5 \times 10^{-8} \times 100 \times 60 \times 60 \times 10 = 0.34\ \text{s}$

Distance traveled by Scud = $(0.34\ \text{s}) \times (1676\ \text{m/s}) \cong 570\ \text{m}$

This put the Scud outside the Patriot's "range gate" Ironically, the fact that the bad time calculation had been improved in some (but not all) code parts contributed to the problem, since it meant that inaccuracies did not cancel out

15

Consequences of Bad Approximations

Example: Explosion of Ariane Rocket (1996 June 4)

Source <http://www.math.psu.edu/dna/455.f96/disasters.html>

Unmanned Ariane 5 rocket launched by the European Space Agency veered off its flight path, broke up, and exploded only 30 seconds after lift-off (altitude of 3700 m)

The \$500 million rocket (with cargo) was on its 1st voyage after a decade of development costing \$7 billion

Cause: "software error in the inertial reference system"

Specifics of the problem: a 64 bit floating point number relating to the horizontal velocity of the rocket was being converted to a 16 bit signed integer

An SRI* software exception arose during conversion because the 64-bit floating point number had a value greater than what could be represented by a 16-bit signed integer (max 32 767)

16

Pentium Bug

October 1994

Thomas Nicely, Lynchburg College, Virginia
finds an error in his computer calculations, and traces
it back to the Pentium processor

November 7, 1994

First press announcement, *Electronic Engineering Times*

Late 1994

Tim Coe, Vitesse Semiconductor
presents an example with the worst-case error

$$c = 4\ 195\ 835/3\ 145\ 727$$

Pentium = 1.333 **739 06...**

Correct result = 1.333 **820 44...**

17

Pentium Bug

Intel admits “subtle flaw”

November 30, 1994

Intel’s white paper about the bug and its possible consequences

Intel - average spreadsheet user affected once in **27,000 years**

IBM - average spreadsheet user affected once every **24 days**

Replacements based on customer needs

December 20, 1994

Announcement of no-question-asked replacements

18

Pentium Bug

Error traced back to the look-up table used by the radix-4 SRT division algorithm

2048 cells, 1066 non-zero values $\{-2, -1, 1, 2\}$

5 non-zero values not downloaded correctly to the lookup table due to an error in the C script

19

Number System Representations

ECE 645 – Computer Arithmetic

Ancient Codes for Numbers—Egyptian

- Egyptian
 - ~4000 BC
 - “Sum of Symbols”
 - $| = 1$ $\frown = 10$ $\text{𐦩} = 100$

$$|||| \frown \frown \frown = (34)_{10}$$

21

Position Codes for Numbers

- Babylonians
 - Positional system (**position has meaning**)
 - 2000 BC
 - Radix 60
 - $\text{𐎶} = 1$ $\text{𐎵} = 10$

22

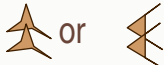

Babylonian Example

1×60^2 $+ 20 \times 60^1$ $+ 56 \times 60^0$

$= (4,856)_{10}$

23

Positional Code with Zero

- Zero Represented by Space
 - Partial solution
 - What about trailing zeros?
- Babylonians Introduced New Symbol
 -  or 
 - 4th to 1st Century BC
- Zero Allows Representation of Fractions
 - Fractions started with zero

24

Mixed System

- Roman Numerals
 - Sum of all symbols
 - I=1 V=5 X=10 L=50 C=100 D=500 M=1000
 - Difficult to do arithmetic
 - e.g.,

MCDXLVII

– IX

?

25

Hindu-Arabic Numeral System

Brahmi numerals, India, 400 BC-400 AD

1	2	3	4	5	6	7	8	9
–	=	≡	+	h	q	7	5	7

Anciens Caractères Arithmétiques.

1. <i>Notes de Boece.</i>	{	1	σ	u	φ	Ϸ	L	.1	δ	9.
2. <i>De Placide.</i>	{	1	μ	ω	ε	ϣ	ν	λ	9	10
3. <i>Indiens à l'Alphabet.</i>	{	1	ρ	μ	ε	ϣ	ν	λ	9	10
4. <i>Indiens de Boece.</i>	{	1	τ	3	2	ϣ	6	λ	8	9
5. <i>De Roger Bacon.</i>	{	1	7	3	2	ϣ	6	λ	8	9
6. <i>Des Indiens Modernes.</i>	{	9	2	ε	ϣ	γ	3	9	τ	9
7. <i>Chiffres Modernes.</i>	{	1	2	3	4	5	6	7	8	9
8. <i>Nombre à l'Alphabet.</i>	{	1	λ	ε	ϣ	ν	λ	ε	ν	μ

Evolution of numerals in early Europe

26

Positional Code Decimal System

- Documented in the 9th century
- Position of coefficient determines its value
- Coefficient in position is multiplied by radix (10) raised to the power determined by its position, *e.g.*,

$$4 * 10^3 + 8 * 10^2 + 5 * 10^1 + 6 * 10^0 = (4,856)_{10}$$

27

Migration of Positional Code

- ~750 AD
 - Zero spread from India to Arabic countries
- ~1250 AD
 - Zero spread to Europe
- Importance of Zero
 - Ease of arithmetic which leads to improved commerce

28

Binary Number System

- Binary
 - Positional number system
 - Two symbols, $B = \{ 0, 1 \}$
 - Easily implemented using switches
 - Easy to implement in electronic circuitry
 - Algebra invented by George Boole (1815-1864)
allows easy manipulation of symbols

$$(0101)_2 = 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = (5)_{10}$$

29

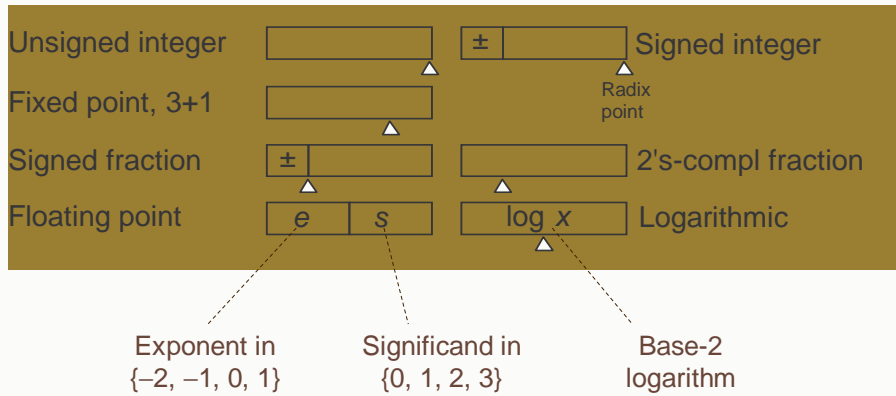
Modern Arithmetic and Number Systems

- Modern number systems used in digital arithmetic can be broadly classified as:
 - **Fixed-point** number representation systems
 - Integers $I = \{-N, \dots, N\}$
 - Rational numbers of the form $x = a/2^f$, a is an integer, f is a positive integer
 - **Floating-point** number representation systems
 - $x * b^E$, where x is a rational number, b the integer base, and E the exponent
- Mainly focus on fixed-point number representation systems in this course
- Note that all digital numbers are eventually coded in bits $\{0,1\}$ on a computer

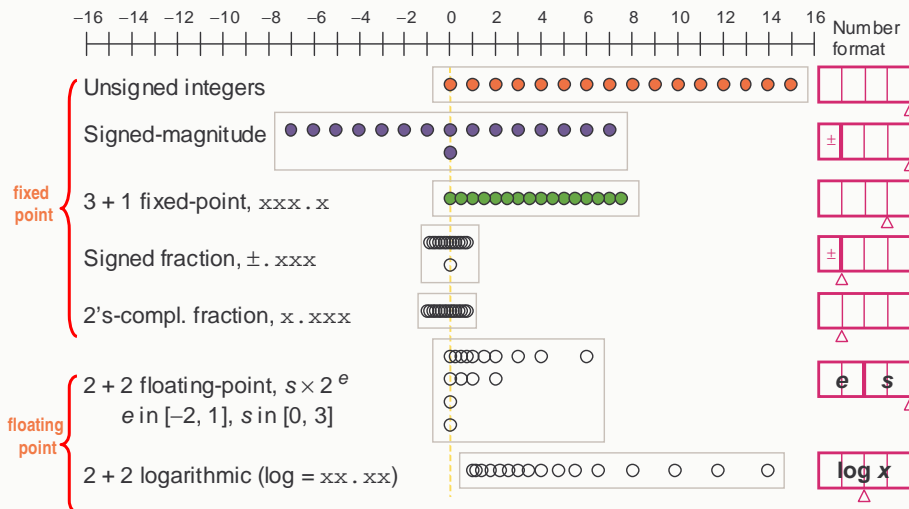
30

Encoding Numbers in 4-Bits

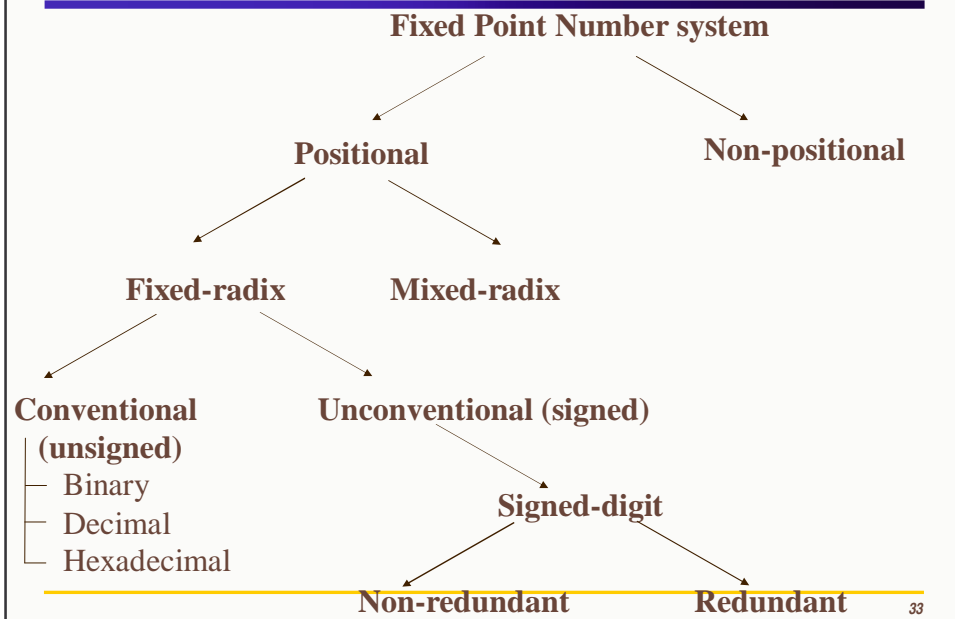
Some 4-bit number representation formats



Encoding Numbers in 4-Bits



Fixed-Point Number Systems



33

Classification of Number Systems

Positional

$$X = \sum_{i=-l}^{k-1} x_i \cdot w_i \quad w_i - \text{weight of the digit } x_i$$

Fixed-radix

$$X = \sum_{i=-l}^{k-1} x_i \cdot r^i \quad r - \text{radix of the number system}$$

Conventional fixed-radix (unsigned)

$$X = \sum_{i=-l}^{k-1} x_i \cdot r^i \quad \begin{array}{l} r \text{ integer, } r > 0 \\ x_i \in \{0, 1, \dots, r-1\} \end{array}$$

34

Classification of Number Systems Cont'd

Unconventional fixed-radix (signed)

$$X = \sum_{i=-l}^{k-1} x_i \cdot r^i \quad x_i \in \{-\alpha, \dots, \beta\}$$

Signed-digit $\alpha > 0 \Rightarrow$ negative digits

Non-redundant number of digits = $\alpha + \beta + 1 \leq r$

Redundant number of digits = $\alpha + \beta + 1 > r$

Fixed-Point Radix Point (a.k.a decimal point)

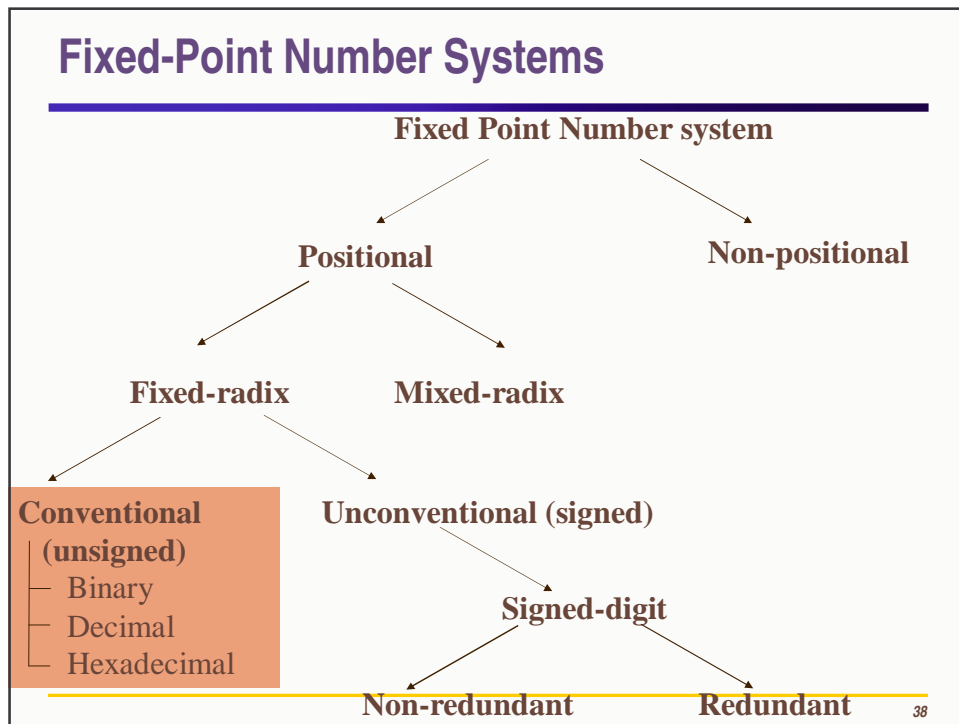
Integral and fractional part

$$X = \underbrace{x_{k-1} x_{k-2} \dots x_1 x_0}_{\text{Integral part}} \cdot \underbrace{x_{-1} x_{-2} \dots x_{-l}}_{\text{Fractional part}}$$

Integral part *Fractional part*
(whole part)

Radix point

- NOT stored in the register
- understood to be in a fixed position
- called decimal point, binary point, etc.



Unsigned versus Signed Representations

- Unsigned range is from 0 to Xmax
 - Example: $(0)_{10}$ to $(15)_{10}$
- Signed range is from Xmin to Xmax
 - Example: $(-8)_{10}$ to $(7)_{10}$

39

Range of Numbers

Number system	X_{\min}	X_{\max}
Decimal $X = (x_{k-1} x_{k-2} \dots x_1 x_0.x_{-1} \dots x_{-l})_{10}$	0	$10^k - 10^{-l}$
Binary $X = (x_{k-1} x_{k-2} \dots x_1 x_0.x_{-1} \dots x_{-l})_2$	0	$2^k - 2^{-l}$
Generic conventional fixed-radix $X = (x_{k-1} x_{k-2} \dots x_1 x_0.x_{-1} \dots x_{-l})_r$	0	$r^k - r^{-l}$

Notation: $ulp = r^{-l}$ unit in the least significant position
 unit in the last position

40

Number of digits

Number system	Number of digits in the integral part necessary to cover the range $0..X_{\max}$
Decimal	$k = \lfloor \log_{10} X_{\max} \rfloor + 1 =$ $= \lceil \log_{10} (X_{\max} + 1) \rceil$
Binary	$k = \lfloor \log_2 X_{\max} \rfloor + 1 =$ $= \lceil \log_2 (X_{\max} + 1) \rceil$
Generic conventional fixed-radix	$k = \lfloor \log_r X_{\max} \rfloor + 1 =$ $= \lceil \log_r (X_{\max} + 1) \rceil$

41

Radix Conversion

Whole part Fractional part

$$\begin{aligned}
 u &= W . V \\
 &= (x_{k-1}x_{k-2} \dots x_1x_0 . x_{-1}x_{-2} \dots x_{-l})_r && \text{Old} \\
 &= (X_{K-1}X_{K-2} \dots X_1X_0 . X_{-1}X_{-2} \dots X_{-L})_R && \text{New}
 \end{aligned}$$

Example: $(31)_{\text{eight}} = (25)_{\text{ten}}$

Two methods:

Option 1) Radix conversion, using arithmetic in the old radix r
*Convenient when converting **from** $r = 10$ or familiar radix*

Option 2) Radix conversion, using arithmetic in the new radix R
*Convenient when converting **to** $R = 10$ or familiar radix*

Option 1: Arithmetic in old radix r

Converting whole part w : Repeatedly divide by five	$(105)_{\text{ten}} = (?)_{\text{five}}$	Quotient	Remainder
		105	0
		21	1
		4	4
		0	

Therefore, $(105)_{\text{ten}} = (410)_{\text{five}}$

Converting fractional part v : Repeatedly multiply by five	$(105.486)_{\text{ten}} = (410.?)_{\text{five}}$	Whole Part	Fraction
			.486
		2	.430
		2	.150
		0	.750
		3	.750
		3	.750

Therefore, $(105.486)_{\text{ten}} \cong (410.22033)_{\text{five}}$

Option 2: Arithmetic in new radix R

Converting whole part w : $(22033)_{\text{five}} = (?)_{\text{ten}}$

$$\begin{array}{r}
 (((((2 \times 5) + 2) \times 5 + 0) \times 5 + 3) \times 5 + 3) \times 5 + 3 \\
 \begin{array}{r}
 |-----| : : : : \\
 10 : : : : \\
 |-----| : : : : \\
 12 : : : : \\
 |-----| : : : : \\
 60 : : : : \\
 |-----| : : : : \\
 303 : : : : \\
 |-----| : : : : \\
 1518 : : : :
 \end{array}
 \end{array}$$

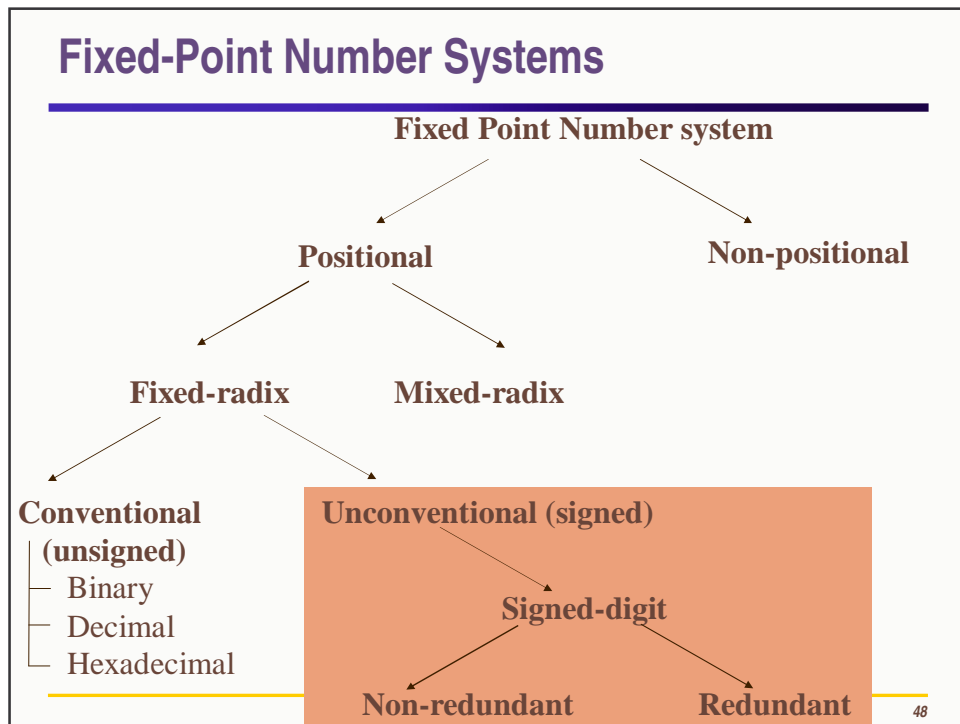
Horner's
rule or
formula

Converting fractional part v : $(410.22033)_{\text{five}} = (105.?)_{\text{ten}}$

$$\begin{array}{l}
 (0.22033)_{\text{five}} \times 5^5 = (22033)_{\text{five}} = (1518)_{\text{ten}} \\
 1518 / 5^5 = 1518 / 3125 = 0.48576
 \end{array}$$

Therefore, $(410.22033)_{\text{five}} = (105.48576)_{\text{ten}}$

Horner's rule is also applicable: Proceed from right to left and use division instead of multiplication (see next slide)



Signed-Digit Representations

- We will discuss the following fixed-radix, unconventional, signed-digit representations
 - 1) Signed-Magnitude
 - redundant
 - 2) Biased (non-redundant)
 - non-redundant
 - 3) Complement
 - A) Radix Complement ($r=2 \rightarrow$ "two's complement")
 - non-redundant
 - B) Digit Complement or Diminished-Radix Complement ($r=2 \rightarrow$ "one's complement")
 - redundant
- Redundant \rightarrow use two representations for the same number
- Non-redundant \rightarrow each representation is a different number

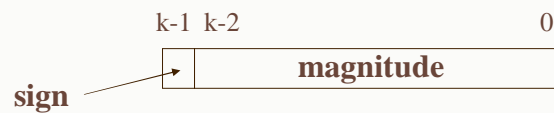
49

	Signed-magnitude	Biased	Two's complement	One's complement
7	0111	1111	0111	0111
6	0110	1110	0110	0110
5	0101	1101	0101	0101
4	0100	1100	0100	0100
3	0011	1011	0011	0011
2	0010	1010	0010	0010
1	0001	1001	0001	0001
0	0000	1000	0000	0000
-0	1000			1111
-1	1001	0111	1111	1110
-2	1010	0110	1110	1101
-3	1011	0101	1101	1100
-4	1100	0100	1100	1011
-5	1101	0011	1011	1010
-6	1110	0010	1010	1001
-7	1111	0001	1001	1000
-8		0000	1000	

50



Signed-Magnitude Representation of Signed Numbers



Advantages:

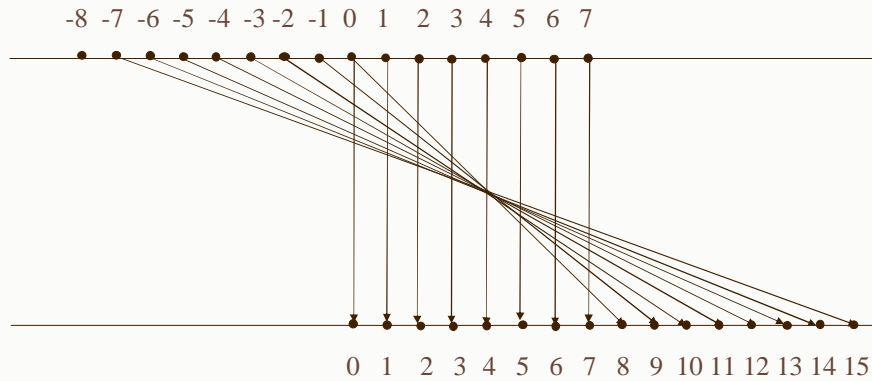
- conceptual simplicity
- symmetric range: $-(2^{k-1}-1) .. -(2^{k-1}-1)$
 - note : zero is redundant (two representations)
- simple negation

Disadvantages:

- addition of numbers with the same sign and with a different sign handled differently

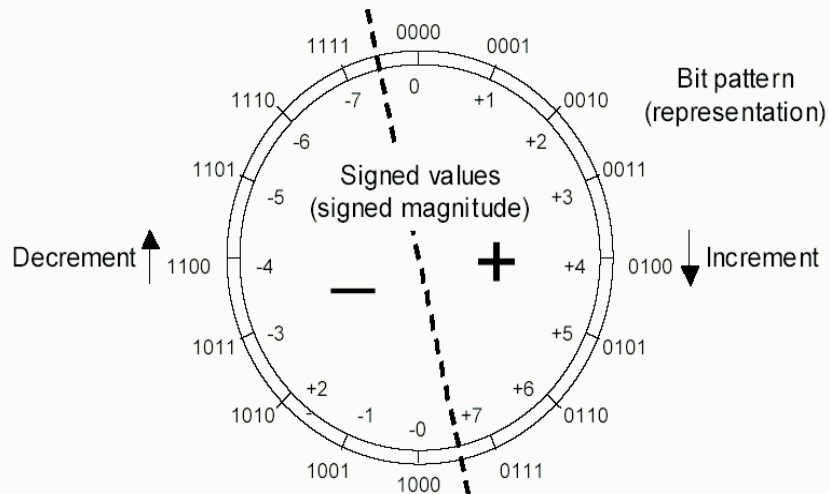
Signed-Magnitude Representation of Signed Numbers

$X > 0$	0	$X < 0$	$k=4$
X	$0, 2^{k-1}$	$ X + 2^{k-1} = -X + 2^{k-1}$	

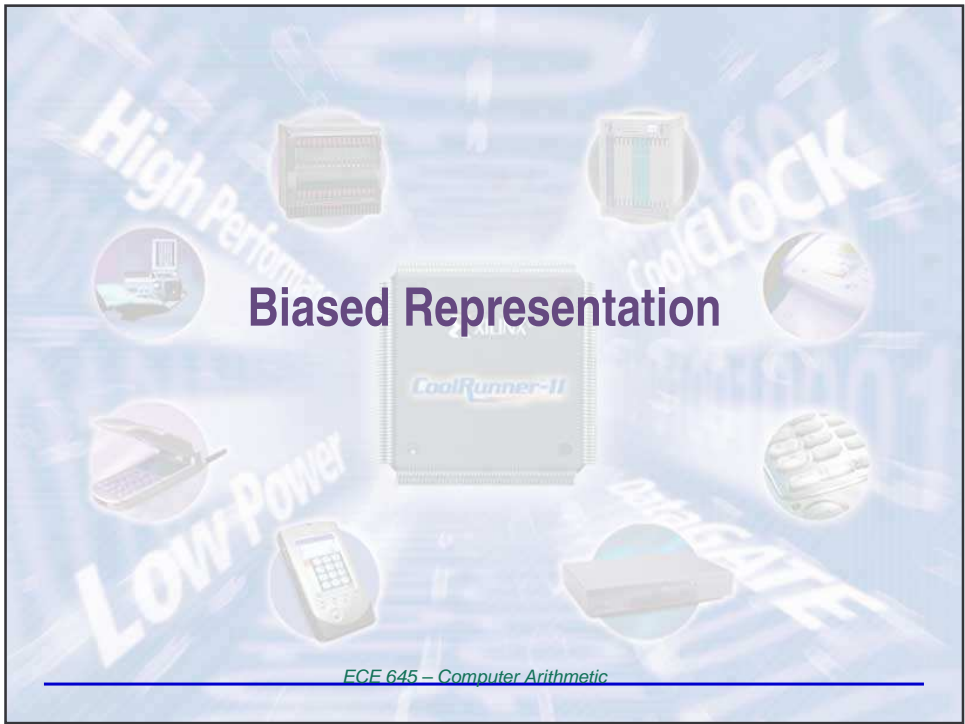


53

4-bit Signed-Magnitude Representation



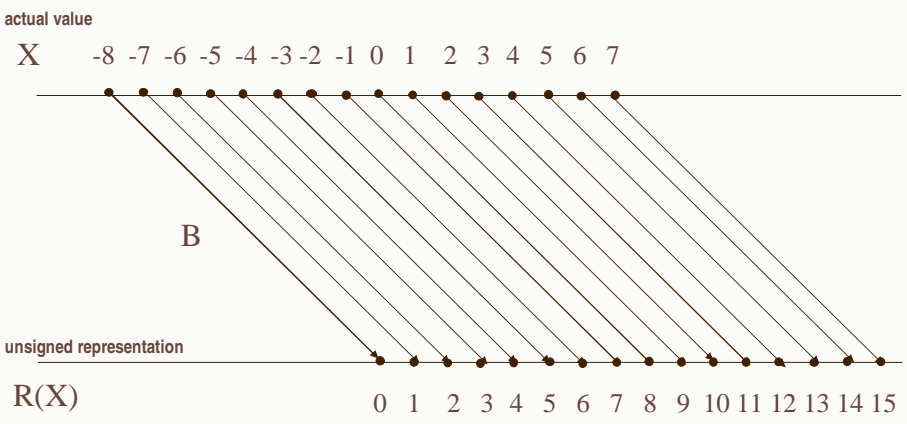
54



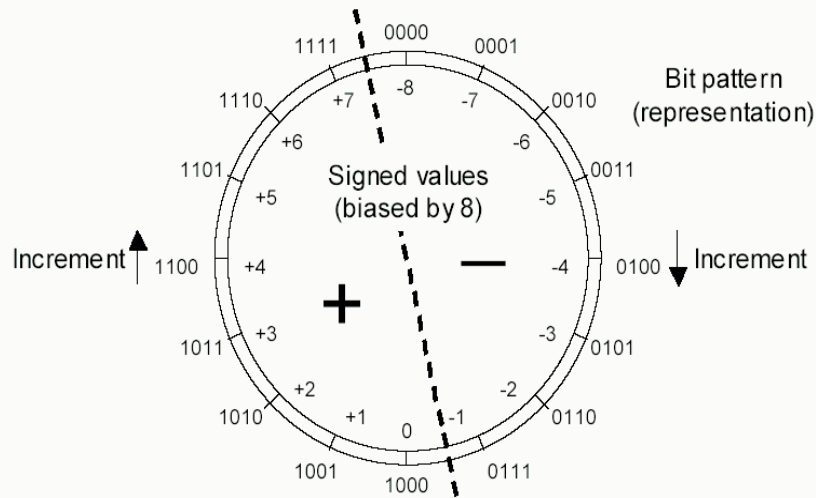
Biased Representation of Signed Numbers

$$R(X) = X + B \qquad B = 2^{k-1}, \quad k=4$$

$$-2^{k-1} \leq X \leq 2^{k-1}-1$$

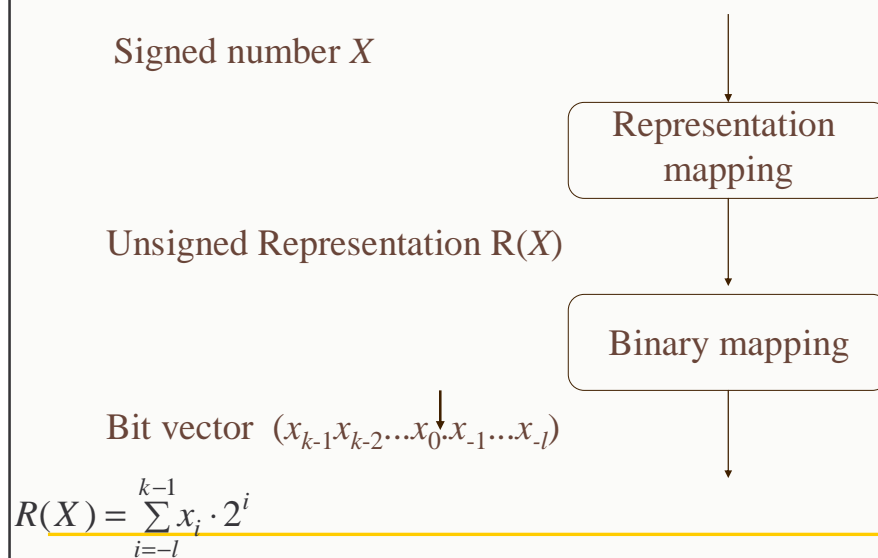


4-bit Biased Representation



57

Biased Representation with Radix 2



58

Biased Representations

- Non-redundant
- Arithmetic is difficult to do because must add or subtract bias from add/subtract operations, since:
 - $x + y + \text{bias} = (x + \text{bias}) + (y + \text{bias}) - \text{bias}$
 - $x - y + \text{bias} = (x + \text{bias}) + (y + \text{bias}) + \text{bias}$

59



Complement Representations with Radix 2

Signed number X

Representation
mapping

Unsigned Representation $R(X)$

Binary mapping

Bit vector $(x_{k-1}x_{k-2}\dots x_0x_{-1}\dots x_{-l})$
$$R(X) = \sum_{i=-l}^{k-1} x_i \cdot 2^i$$

61

Useful Dependencies

$$1 - x_i = \overline{x_i}$$

x_i	$1 - x_i$	$\overline{x_i}$
0	1	1
1	0	0

$$|X| = \begin{cases} X & \text{when } X \geq 0 \\ -X & \text{when } X < 0 \end{cases}$$

62



One's Complement Transformation

For

$$A = \sum_{i=-l}^{k-1} A_i \cdot 2^i \geq 0$$

$$\mathbf{OC(A)} \stackrel{\text{def}}{=} \bar{A} = 2^k - 2^{-l} - A$$

$$\begin{array}{cccccccc}
 & k-1 & k-2 & \dots & 0 & -1 & -2 & \dots & -l \\
 & 1 & 1 & \dots & 1 & \cdot & 1 & 1 & \dots & 1 \\
 - & A_{k-1} & A_{k-2} & \dots & A_0 & \cdot & A_{-1} & A_{-2} & \dots & A_{-l} \\
 \hline
 & \bar{A}_{k-1} & \bar{A}_{k-2} & \dots & \bar{A}_0 & \cdot & \bar{A}_{-1} & \bar{A}_{-2} & \dots & \bar{A}_{-l}
 \end{array}$$

Properties:

$0 \leq OC(A) \leq 2^k - 2^{-l}$

OC(OC(A)) = A

One's Complement Representation of Signed Numbers

For $-(2^{k-1} - 2^{-l}) \leq X \leq 2^{k-1} - 2^{-l}$

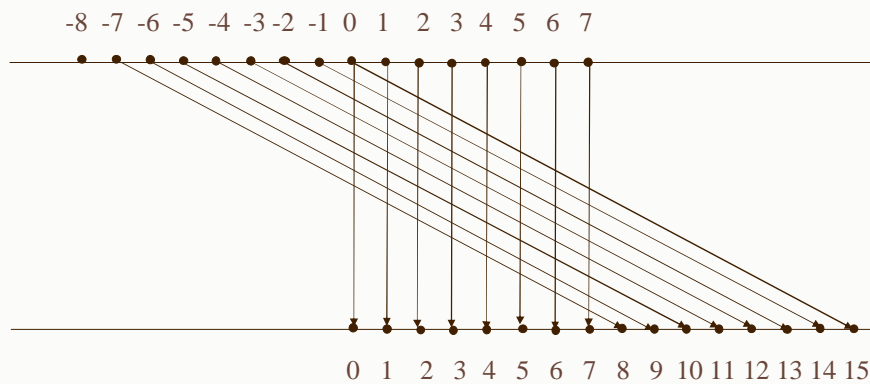
$$\stackrel{\text{def}}{R(X)} = \begin{cases} X & \text{for } X > 0 \\ 0 \text{ or OC}(0) & \text{for } X = 0 \\ \text{OC}(|X|) & \text{for } X < 0 \end{cases}$$

$$0 \leq R(X) \leq 2^k - 2^{-l}$$

65

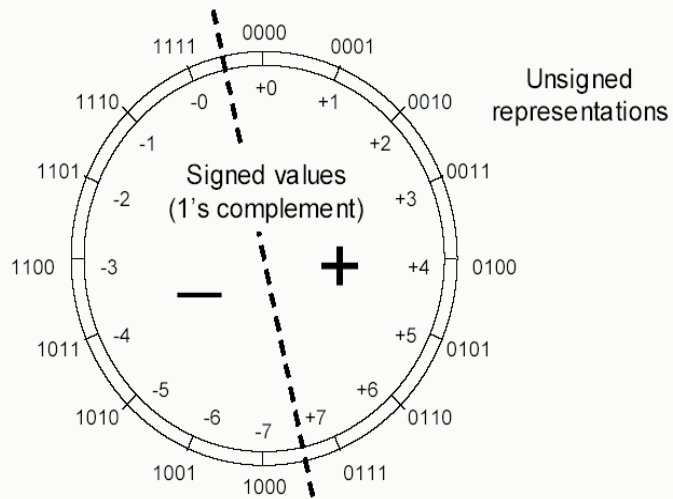
One's Complement Mapping

$X > 0$	0	$X < 0$	$k=4$
X	0, 2^{k-1}	$X + 2^k - 1 = 2^k - 1 - X $	



66

4-bit One's Complement Representation



67

Two's Complement Representation



Two's Complement Transformation (1)

For $A = \sum_{i=-l}^{k-1} A_i \cdot 2^i \geq 0$

$$\text{TC}(A) \stackrel{\text{def}}{=} \begin{cases} \bar{A} + 2^{-l} = 2^k - A & \text{for } A > 0 \\ 0 & \text{for } A = 0 \end{cases}$$

$$\begin{aligned} 2^k - A &= 2^k - A - 2^{-l} + 2^{-l} = \\ &= (2^k - 2^{-l} - A) + 2^{-l} = \bar{A} + 2^{-l} \end{aligned}$$

Properties:

$$0 \leq \text{TC}(A) \leq 2^k - 2^{-l}$$

$$\text{TC}(\text{TC}(A)) = A$$

69

Two's Complement Transformation(2)

For $A = \sum_{i=-l}^{k-1} A_i \cdot 2^i \geq 0$

$$\text{TC}(A) \stackrel{\text{def}}{=} \bar{A} + 2^{-l} \bmod 2^k = 2^k - A \bmod 2^k$$

70

Two's Complement Representation of Signed Numbers

For $-2^{k-1} \leq X \leq 2^{k-1} - 2^{-l}$

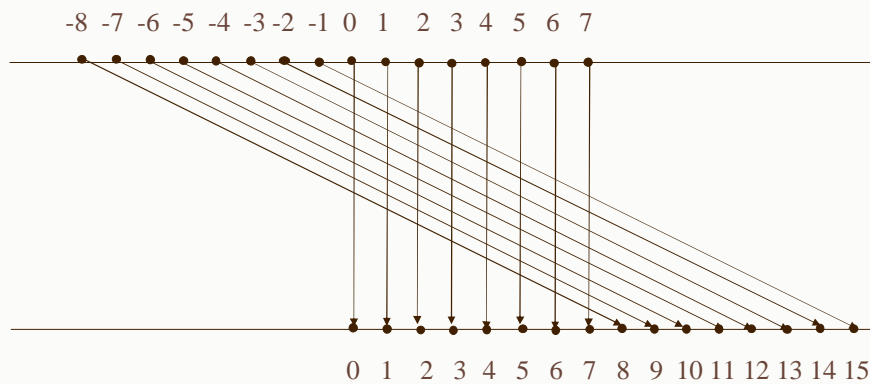
$$R(X) \stackrel{\text{def}}{=} \begin{cases} X & \text{for } X \geq 0 \\ \text{TC}(|X|) & \text{for } X < 0 \end{cases}$$

$$0 \leq R(X) \leq 2^k - 2^{-l}$$

71

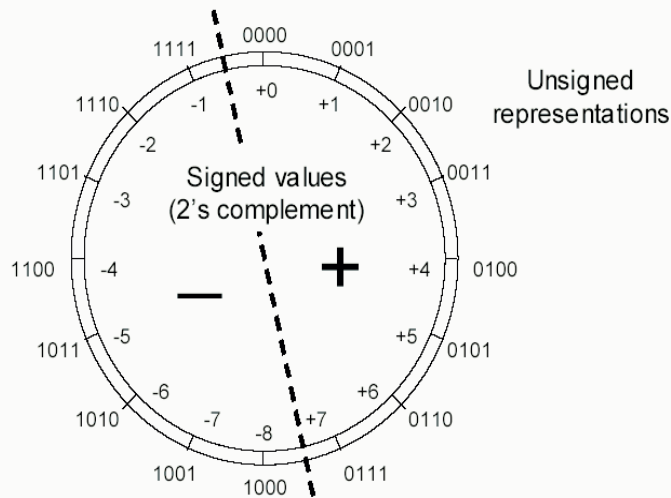
Two's Complement Representation of Signed Integers

$X > 0$	0	$X < 0$	$k=4$
X	0	$X + 2^k = 2^k - X $	



72

4-bit Two's Complement Representation



73

One's Complement versus Two's Complement

Feature	Radix Complement (Two's Complement)	Digit Complement (One's Complement)
Symmetry	Possible for odd r (radices of practical interest are even)	Possible for even r
Unique zero?	Yes	No
Complementation	Complement all digits and add ulp	Complement all digits
Mod-M Addition	Drop carry-out	End-around carry

- Two's complement is easier to implement than one's complement in hardware (addition, subtraction, multiplication), even though it is not symmetric
- In the majority of digital systems, representation is either in:
 - Two's complement → when dealing with positive and negative numbers
 - Unsigned binary → when dealing with non-negative numbers only

74

Out of range flags

Carry flag - C

out-of-range for unsigned numbers

C = 1 if **result > MAX_UNSIGNED** or
result < 0
0 otherwise

where MAX_UNSIGNED = 2^8-1 for 8-bit operands
 $2^{16}-1$ for 16-bit operands

Overflow flag - V

out-of-range for signed numbers

V = 1 if **result > MAX_SIGNED** or
result < MIN_SIGNED
0 otherwise

where MAX_SIGNED = 2^7-1 for 8-bit operands (two's complement)
 $2^{15}-1$ for 16-bit operands (two's complement)
MIN_SIGNED = -2^7 for 8-bit operands (two's complement)
 -2^{15} for 16-bit operands (two's complement)

77

Overflow for Signed Numbers

Indication of overflow

$$\begin{array}{r} \text{Positive} \\ + \text{ Positive} \\ \hline = \text{Negative} \end{array}$$

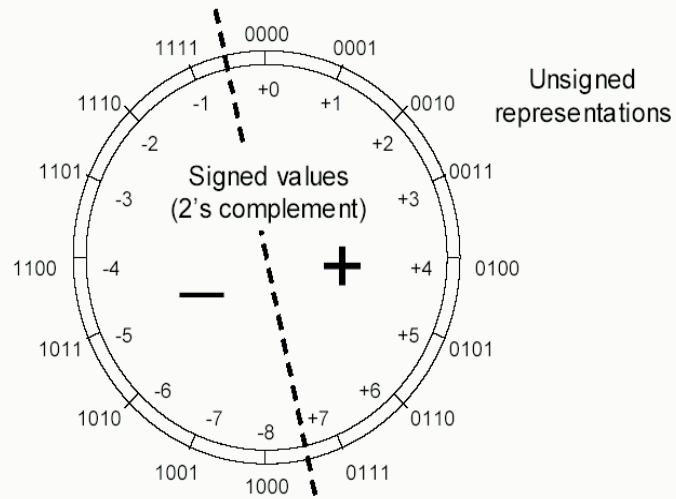
$$\begin{array}{r} \text{Negative} \\ + \text{ Negative} \\ \hline = \text{Positive} \end{array}$$

Formulas

$$\begin{aligned} \text{Overflow}_{2\text{'s complement}} &= \overline{x_{k-1}} \overline{y_{k-1}} s_{k-1} + x_{k-1} y_{k-1} \overline{s_{k-1}} = \\ &= c_k \oplus c_{k-1} \end{aligned}$$

78

4-bit Two's Complement Representation



79

Addition and Subtraction

Two's complement

Numbers of the same sign

$$\begin{array}{r}
 -16 \ 8 \ 4 \ 2 \ 1 \\
 1 \ 1 \ 0 \ 1 \ 1 \quad -5 \\
 1 \ 0 \ 1 \ 1 \ 0 \quad -10 \quad + \\
 \hline
 1 \ 1 \ 0 \ 0 \ 0 \ 1 \quad -15
 \end{array}$$

carry but not overflow

$$\begin{array}{r}
 -16 \ 8 \ 4 \ 2 \ 1 \\
 \boxed{0} \ 0 \ 1 \ 1 \ 1 \quad 7 \\
 \boxed{0} \ 1 \ 0 \ 1 \ 0 \quad 10 \quad + \\
 \hline
 \boxed{0} \ 1 \ 0 \ 0 \ 0 \ 1 \quad -15
 \end{array}$$

no carry but overflow

Numbers of the opposite sign

$$\begin{array}{r}
 -16 \ 8 \ 4 \ 2 \ 1 \\
 0 \ 0 \ 1 \ 0 \ 1 \quad 5 \\
 1 \ 0 \ 1 \ 1 \ 0 \quad -10 \quad + \\
 \hline
 0 \ 1 \ 1 \ 0 \ 1 \ 1 \quad -5
 \end{array}$$

$$\begin{array}{r}
 -16 \ 8 \ 4 \ 2 \ 1 \\
 0 \ 1 \ 0 \ 1 \ 0 \quad 10 \\
 1 \ 1 \ 0 \ 1 \ 1 \quad -5 \quad + \\
 \hline
 1 \ 0 \ 0 \ 1 \ 0 \ 1 \quad 5
 \end{array}$$

carry but not overflow

80

Subtraction

- To subtract $A - B$
 - 1) invert bits of B
 - 2) add up to B
 - 3) perform addition as on previous page: $A + (-B)$
- Example : $0011 - 0001$
$$\begin{array}{r} 0011 + \\ 1110 + \\ 1 \\ \hline 10010 \end{array}$$
- The ulp (LSB) can be added into the carryin part of an adder, saving hardware and making a universal adder/subtractor

81

Two's Complement Adder/Subtractor Architecture

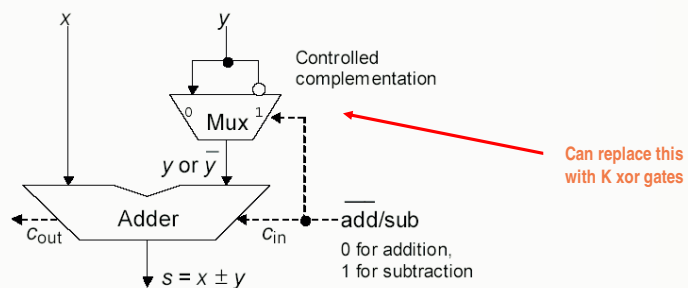


Fig. 2.7 Adder/subtractor architecture for two's-complement numbers.

82

Arithmetic Shift

One's complement

$$\text{Sh.L } \{00101_2 = +5\} = 01010_2 = +10$$

$$\text{Sh.L } \{11010_2 = -5\} = 1010\boxed{1}_2 = -10$$

$$\text{Sh.L } \{01010_2 = +10\} = 10100_2 = -11$$

Shift left may cause overflow

overflow

$$\text{Sh.R } \{00101_2 = +5\} = \boxed{0}0010_2 = +2 \quad \text{rem } 1$$

$$\text{Sh.R } \{11011_2 = -5\} = \boxed{1}1101_2 = -2 \quad \text{rem } -1$$

Shift right requires sign extension

85

Addition and Subtraction

Signed-magnitude

Numbers of the same sign

Numbers of the opposite sign

	sign bit	magnitude	
	↓	↓	
	0	1 0 1 1	11
+	0	0 1 1 0	6
	0	1 0 0 0 1	17

↑
carry = overflow

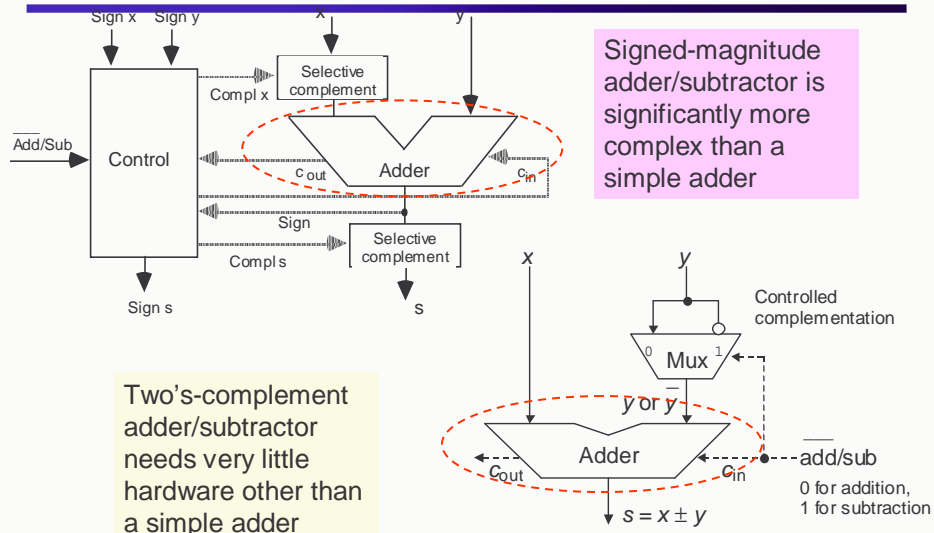
	sign bit	magnitude	
	↓	↓	
	1	1 0 1 1	-11
+	0	0 1 1 0	6

11 > 6

		1 0 1 1	11
-		0 1 1 0	6
	1	0 1 0 1	5

86

Signed-Magnitude Adder



87

Arithmetic with Biased Adders

Addition/subtraction of biased numbers

$$x + y + bias = (x + bias) + (y + bias) - bias$$

$$x - y + bias = (x + bias) - (y + bias) + bias$$

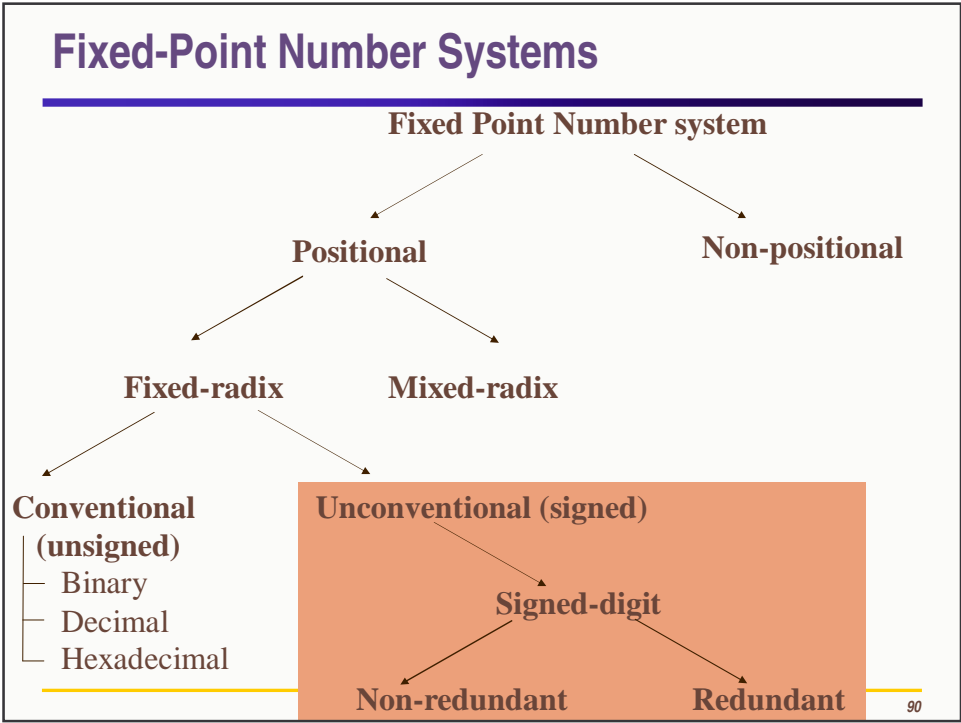
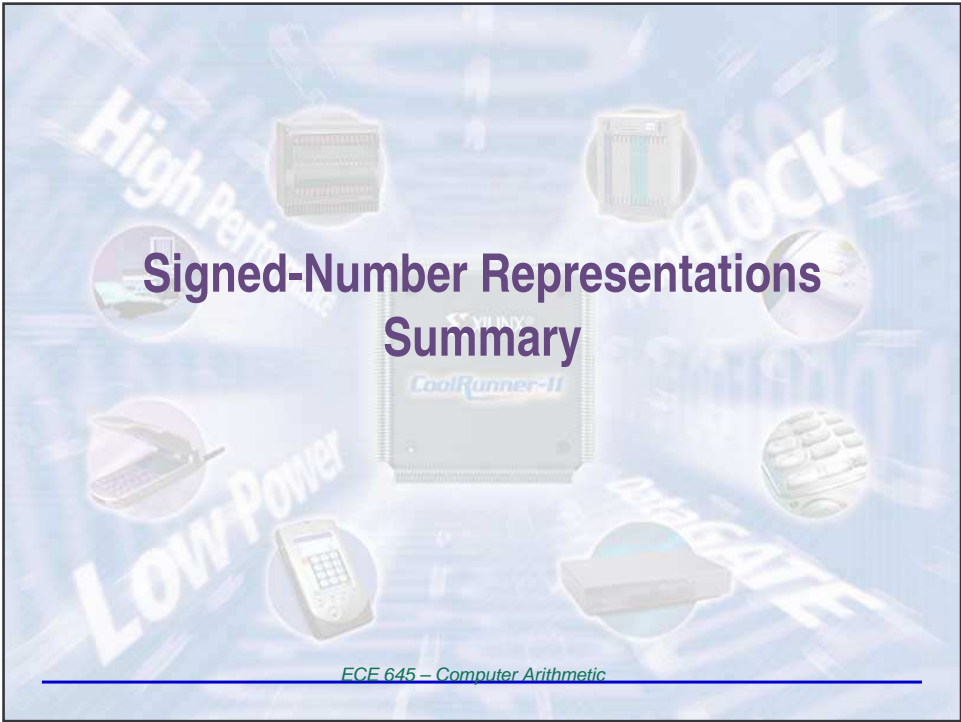
A power-of-2 (or $2^a - 1$) bias simplifies addition/subtraction

Comparison of biased numbers:

Compare like ordinary unsigned numbers
find true difference by ordinary subtraction

We seldom perform arbitrary arithmetic on biased numbers
Main application: Exponent field of floating-point numbers

88



K-bit Signed Binary Numbers

Representation	Representation for $X > 0$	Representation for 0	Representation for $X < 0$
Signed-magnitude	X	0, 2^{k-1}	$2^{k-1} + X $
Biased	$X+B$	B typical $B=2^{k-1}$ or $2^{k-1}-ulp$	$X+B$
Complement	X	0, $M \bmod 2^k$	$M - X = M + X$
Two's complement	X	0	$2^k - X = \overline{ X } + ulp$
One's complement	X	0, $2^k - ulp$	$2^k - ulp - X = \overline{ X }$

91

Value of the Number in Signed Representations

Representation	Value of $(x_{k-1} x_{k-2} \dots x_1 x_0 x_{-1} \dots x_{-l})$
Signed-magnitude	$X = (-1)^{x_{k-1}} \sum_{i=-l}^{k-2} x_i \cdot 2^i$
Biased	$X = \sum_{i=-l}^{k-1} x_i \cdot 2^i - B$
Two's complement	$X = -x_{k-1} 2^{k-1} + \sum_{i=-l}^{k-2} x_i \cdot 2^i$
One's complement	$X = -x_{k-1} (2^{k-1} - ulp) + \sum_{i=-l}^{k-2} x_i \cdot 2^i$

92

Extending Numbers of Bits of a Signed Number

$X \quad x_{k-1} \ x_{k-2} \ \dots \ x_1 \ x_0 \cdot x_{-1} \ x_{-2} \ \dots \ x_{-l}$



$Y \quad y_{k'-1} \ y_{k'-2} \ \dots \ y_k \ y_{k-1} \ y_{k-2} \ \dots \ y_1 \ y_0 \cdot y_{-1} \ y_{-2} \ \dots \ y_{-l} \ y_{-(l+1)} \ \dots \ y_{-l'}$

signed-magnitude

$x_{k-1} \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ x_{k-2} \ \dots \ x_1 \ x_0 \cdot x_{-1} \ x_{-2} \ \dots \ x_{-l} \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$

two's complement

$x_{k-1} \ x_{k-1} \ x_{k-1} \ \dots \ x_{k-1} \ x_{k-2} \ \dots \ x_1 \ x_0 \cdot x_{-1} \ x_{-2} \ \dots \ x_{-l} \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$

one's complement

$x_{k-1} \ x_{k-1} \ x_{k-1} \ \dots \ x_{k-1} \ x_{k-2} \ \dots \ x_1 \ x_0 \cdot x_{-1} \ x_{-2} \ \dots \ x_{-l} \ x_{k-1} \ \dots \ x_{k-1}$

biased

$\overline{x_{k-1}} \ \overline{x_{k-1}} \ \dots \ \overline{x_{k-1}} \ x_{k-2} \ \dots \ x_1 \ x_0 \cdot x_{-1} \ x_{-2} \ \dots \ x_{-l} \ 0 \ 0 \ 0 \ 0 \ 0 \ 0$

93

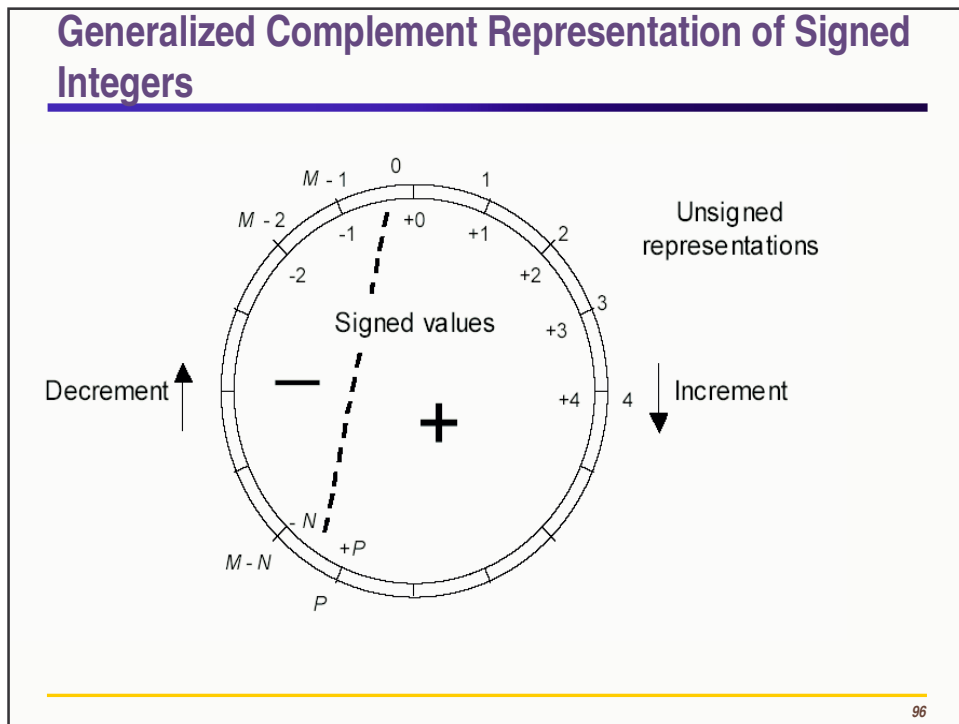
Signed Numbers: Usability

- The vast majority of digital systems use either two's complement or unsigned binary
 - Two's complement → when dealing with positive and negative numbers
 - VHDL: signed data type
 - 0000 → 0
 - 0111 → 7
 - 1000 → -8
 - 1111 → -1
 - Unsigned binary → when dealing with non-negative numbers only
 - VHDL: unsigned data type
 - 0000 → 0
 - 0111 → 7
 - 1000 → 8
 - 1111 → 15

94

Generalized Complement Representation

ECE 645 - Computer Arithmetic



Generalized Complement Representation of Signed Integers

Table 2.1 Addition in a complement number system with complementation constant M and range $[-N, +P]$

Desired operation	Computation to be performed mod M	Correct result with no overflow	Overflow condition
$(+x) + (+y)$	$x + y$	$x + y$	$x + y > P$
$(+x) + (-y)$	$x + (M - y)$	$x - y$ if $y \leq x$ $M - (y - x)$ if $y > x$	N/A
$(-x) + (+y)$	$(M - x) + y$	$y - x$ if $x \leq y$ $M - (x - y)$ if $x > y$	N/A
$(-x) + (-y)$	$(M - x) + (M - y)$	$M - (x + y)$	$x + y > N$