Lecture 1: Introduction and Number Representations

ECE 645—Computer Arithmetic
1/22/08

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

Syllabus and Course Objectives
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

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
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

<table>
<thead>
<tr>
<th>Environment</th>
<th>Simulation</th>
<th>Synthesis</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aldec Active-HDL 7.2 SP2</td>
<td>Aldec Active-HDL 7.2 SP2</td>
<td>Synplicity Synplify Pro 8.6.2</td>
<td>Xilinx ISE Foundation 9.1 SP3</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>Xilinx ISE Foundation 9.1 SP3</td>
<td>Mentor Graphics Modelsim SE 6.3a</td>
<td>Synplicity Synplify Pro 8.6.2</td>
<td>Xilinx ISE Foundation 9.1 SP3</td>
</tr>
<tr>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

- 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:
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

What is Computer Arithmetic?

**Hardware (our focus in this class)**

- Design of efficient digital circuits for primitive and other arithmetic operations such as +, −, ×, ÷, √, log, sin, cos
- **Issues:** Algorithms, Error analysis, Speed/cost trade-offs, Hardware implementation, Testing, verification

**Software**

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

**Issues:** Algorithms, Error analysis, Computational complexity, Programming, Testing, verification

**General-purpose**

- Flexible data paths
- Fast primitive operations like +, −, ×, ÷, √
- Benchmarking

**Special-purpose**

- Tailored to applications like: Digital filtering, Image processing, Radar tracking

- From Parhami’s slides
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

Calculators

\[ u = \sqrt{\sqrt{\sqrt{\ldots\sqrt{2}}}} = 1.000\,677\,131 \]
\[ 10 \text{ times} \]
\[ x = (((u^2)^2)\ldots)^2 = 1.999\,999\,963 \]
\[ 10 \text{ times} \]
\[ x' = u^{1024} = 1.999\,999\,973 \]

\[ v = 2^{1/1024} = 1.000\,677\,131 \]

\[ y = (((v^2)^2)\ldots)^2 = 1.999\,999\,983 \]
\[ 10 \text{ times} \]
\[ y' = v^{1024} = 1.999\,999\,994 \]

Hidden digits in the internal representation of numbers
Different algorithms give slightly different results
Very good accuracy
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 1001 100 (chopped to 24 b)

Error \( \approx 0.1100 \times 2^{-23} = 9.5 \times 10^{-8} \)

Error in 100-hr operation period
\( \approx 9.5 \times 10^{-8} \times 100 \times 60 \times 60 \times 10 = 0.34 \) s

Distance traveled by Scud = \( (0.34 \) s) \times (1676 \text{ m/s}) \approx 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.

---

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).
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

\[ c = \frac{4\,195\,835}{3\,145\,727} \]

Pentium = 1.333 739 06...
Correct result = 1.333 820 44...

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
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
Ancient Codes for Numbers—Egyptian

- Egyptian
  - ~4000 BC
  - “Sum of Symbols”
    \[
    \begin{align*}
    \text{I} &= 1 \\
    \text{V} &= 5 \\
    \text{X} &= 10 \\
    \text{L} &= 50 \\
    \text{C} &= 100
    \end{align*}
    \]
    \[
    \text{|||||| ||} = 34 \quad (34)_{10}
    \]

Position Codes for Numbers

- Babylonians
  - Positional system (position has meaning)
  - 2000 BC
  - Radix 60
  - \[
  \begin{align*}
  \text{I} &= 1 \\
  \text{X} &= 10
  \end{align*}
  \]
Babylonian Example

\[ 1 \times 60^2 + 20 \times 60^1 + 56 \times 60^0 = (4,856)_{10} \]

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
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.,

\[
\text{MCDXLVII} \\
- \text{IX} \\
= \text{CDXLVII}
\]

Hindu-Arabic Numeral System

Brahmi numerals, India, 400 BC-400 AD

Evolution of numerals in early Europe
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.,

\[ 4 \times 10^3 + 8 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 = (4,856)_{10} \]

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
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
    \[
    (0101)_2 = 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}
    \]

Modern Arithmetic and Number Systems

- Modern number systems used in digital arithmetic can be broadly classified as:
  - Fixed-point number representation systems
    - Integers \( I = \{-N, \ldots, N\} \)
    - Rational numbers of the form \( x = a/2^f \), \( a \) is an integer, \( f \) is a positive integer
  - Floating-point number representation systems
    - \( x \times b^E \), 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
Some 4-bit number representation formats

- **Unsigned integer**
- **Signed integer**
- **Fixed point, 3+1**
- **Signed fraction**
- **2's-compl fraction**
- **Floating point**
- **Logarithmic**

- **Exponent in** \{-2, -1, 0, 1\}
- **Significand in** \{0, 1, 2, 3\}
- **Base-2 logarithm**

Encoding Numbers in 4-Bits

- **Unsigned integers**
- **Signed-magnitude**
- **3 + 1 fixed-point, xxx.x**
- **Signed fraction, ±.xxx**
- **2’s-compl. fraction, x.xxx**
- **2 + 2 floating-point, s \times 2^e**  
  \(e\) in \{-2, 1\}, \(s\) in \{0, 3\}
- **2 + 2 logarithmic (log = xx.xxx)**

Number format

- **Log x**
Fixed-Point Number Systems

Classification of Number Systems

Positional

\[ X = \sum_{i=-l}^{k-1} x_i \cdot w_i \]

- \( w_i \) - weight of the digit \( x_i \)

Fixed-radix

\[ X = \sum_{i=-l}^{k-1} x_i \cdot r^i \]

- \( r \) - radix of the number system

Conventional fixed-radix (unsigned)

\[ X = \sum_{i=-l}^{k-1} x_i \cdot r^i \]

- \( r \) integer, \( r > 0 \)
- \( x_i \in \{0, 1, \ldots, r-1\} \)
Unconventional fixed-radix (signed)

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

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

\[ X = x_{k-1} x_{k-2} \ldots x_1 x_0 \cdot x_{-1} x_{-2} \ldots x_{-l} \]

\( \text{Integral part} \quad \text{Fractional part} \quad (\text{whole part}) \)

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

Fixed-Point Number Systems

- Fixed Point Number system
  - Positional
    - Fixed-radix
  - Non-positional
    - Mixed-radix

- Conventional (unsigned)
  - Binary
  - Decimal
  - Hexadecimal

- Unconventional (signed)
  - Signed-digit

- Non-redundant
- Redundant
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}

Range of Numbers

<table>
<thead>
<tr>
<th>Number system</th>
<th>$X_{\text{min}}$</th>
<th>$X_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>0</td>
<td>$10^k - 10^{-l}$</td>
</tr>
<tr>
<td>$X = (x_{k-1} x_{k-2} \ldots x_1 x_0 - x_{-1} \ldots x_{-l})_{10}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td>0</td>
<td>$2^k - 2^{-l}$</td>
</tr>
<tr>
<td>$X = (x_{k-1} x_{k-2} \ldots x_1 x_0 - x_{-1} \ldots x_{-l})_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generic conventional fixed-radix</td>
<td>0</td>
<td>$r^k - r^{-l}$</td>
</tr>
<tr>
<td>$X = (x_{k-1} x_{k-2} \ldots x_1 x_0 - x_{-1} \ldots x_{-l})_r$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Number system</th>
<th>Number of digits in the integral part necessary to cover the range 0..&lt;(X_{\text{max}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>(k = \lfloor \log_{10} X_{\text{max}} \rfloor + 1 = \left\lfloor \log_{10} (X_{\text{max}} + 1) \right\rfloor )</td>
</tr>
<tr>
<td>Binary</td>
<td>(k = \lfloor \log_{2} X_{\text{max}} \rfloor + 1 = \left\lfloor \log_{2} (X_{\text{max}} + 1) \right\rfloor )</td>
</tr>
<tr>
<td>Generic conventional fixed-radix</td>
<td>(k = \lfloor \log_{r} X_{\text{max}} \rfloor + 1 = \left\lfloor \log_{r} (X_{\text{max}} + 1) \right\rfloor )</td>
</tr>
</tbody>
</table>

### Radix Conversion

Whole part \(u = w \cdot v\)

\[
\begin{align*}
  u &= (x_{k-1}x_{k-2} \ldots x_1x_0 \cdot x_{-1}x_{-2} \ldots x_{-l})_r \\
  &= (X_{K-1}X_{K-2} \ldots X_1X_0 \cdot X_{-1}X_{-2} \ldots X_{-L})_R
\end{align*}
\]

Old \(\text{New}\)

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*

---

From: Parhami, Computer Arithmetic: Algorithms and Hardware Design
Option 1: Arithmetic in old radix \( r \)

Converting whole part \( w \):

\[ (105)_{10} = (?)_{5} \]

\begin{array}{c|c|c}
Quotient & Remainder \\
\hline
105 & 0 \\
21 & 1 \\
4 & 4 \\
0 & \\
\end{array}

Therefore, \( (105)_{10} = (410)_{5} \)

Converting fractional part \( v \):

\[ (105.486)_{10} = (410.)_{5} \]

\begin{array}{c|c|c}
Whole Part & Fraction \\
\hline
2 & .486 \\
2 & .430 \\
0 & .150 \\
3 & .750 \\
3 & .750 \\
\end{array}

Therefore, \( (105.486)_{10} \approx (410.22033)_{5} \)

From: Parhami, Computer Arithmetic: Algorithms and Hardware Design

Option 2: Arithmetic in new radix \( R \)

Converting whole part \( w \):

\[ (22033)_{5} = (?)_{10} \]

\[ (((2 \times 5) + 2) \times 5 + 0) \times 5 + 3) \times 5 + 3 \]

\[ \frac{10}{12} \frac{60}{303} \]

Horner's rule or formula

\[ 1518 \]

Converting fractional part \( v \):

\[ (410.22033)_{5} = (105.?)_{10} \]

\[ (0.22033)_{5} \times 5^5 = (22033)_{5} = (1518)_{10} \]

\[ 1518 / 5^5 = 1518 / 3125 = 0.48576 \]

Therefore, \( (410.22033)_{5} = (105.48576)_{10} \)

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

From: Parhami, Computer Arithmetic: Algorithms and Hardware Design
Option 2 cont'd: Horner's rule for fractions

Converting fractional part \( v \):

\[
(0.22033)_{\text{live}} = (?)_{\text{ten}}
\]

\[
\left( \frac{3}{5} + 3 \right) / 5 + 0 / 5 + 2 / 5 + 2 / 5
\]

\[
\begin{array}{c|c|c|c}
0.6 & 3.6 & 0.72 & 2.144 \\
\hline
& & & 2.4288
\end{array}
\]

\[
\begin{array}{c|c}
2.4288 & 0.48576 \\
\hline
\end{array}
\]

From: Parhami, Computer Arithmetic: Algorithms and Hardware Design

Radix Conversion Shortcut for \( r=b^g, R=b^G \)

\[
r=b^g \rightarrow b \rightarrow R=b^G
\]

\[
4=2^2 \rightarrow 2 \rightarrow 8=2^3
\]

\[
(2301.302)_{4} = (10 \bar{1} 10 \bar{0} 01 \bar{1} 10 \bar{0} 10)_{2} = (261.62)_{8}
\]

- Trick here is to first convert to a number in radix \( b \), then to \( R \)
- Cluster in groups of 3 (because \( 2^3 = 8 \)) moving away from binary point
Fixed-Radix Signed Representations

Fixed-Point Number Systems

- Fixed Point Number system
  - Positional
    - Fixed-radix
      - Conventional (unsigned)
        - Binary
        - Decimal
        - Hexadecimal
      - Unconventional (signed)
        - Signed-digit
          - Non-redundant
          - Redundant
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 \(r=2\) → "two’s complement"
      - non-redundant
    - B) Digit Complement or Diminished-Radix Complement \(r=2\) → "one’s complement"
      - redundant

- Redundant → use two representations for the same number
- Non-redundant → each representation is a different number
Signed-Magnitude Representation of Signed Numbers

Advantages:

- conceptual simplicity
- symmetric range: \(-(2^{k-1}-1) \ldots -(2^{k-1}-1)\)
  - 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

<table>
<thead>
<tr>
<th>$X &gt; 0$</th>
<th>0</th>
<th>$X &lt; 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$0$, $2^{k-1}$</td>
<td>$</td>
</tr>
</tbody>
</table>

$k = 4$

<table>
<thead>
<tr>
<th>$X$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

4-bit Signed-Magnitude Representation

Signed values (signed magnitude)

-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7

Bit pattern (representation)

Increment

Decrement
Biased Representation

Biased Representation of Signed Numbers

\[ R(X) = X + B \]

\[ B = 2^{k-1}, \ k=4 \]

\[ -2^{k-1} \leq X \leq 2^{k-1} - 1 \]
4-bit Biased Representation

Bit pattern (representation)

Signed values (biased by 8)

Increment

Biased Representation with Radix 2

Signed number $X$

Representation mapping

Unsigned Representation $R(X)$

Binary mapping

Bit vector $(x_{k-1}x_{k-2}...x_0.x_{-1}...x_{-l})$

$$R(X) = \sum_{i=-l}^{k-1} x_i \cdot 2^i$$
Biased Representations

- Non-redundant
- Arithmetic is difficult to do because must add or subtract bias from add/subtract operations, since:
  - $x + y + \text{bias} = (x + \text{bias}) + (y + \text{bias}) - \text{bias}$
  - $x - y + \text{bias} = (x + \text{bias}) + (y + \text{bias}) + \text{bias}$
**Complement Representations with Radix 2**

Signed number $X$

Representation mapping

Unsigned Representation $R(X)$

Binary mapping

Bit vector $(x_{k-1}x_{k-2}...x_0\cdot x_{-1}...x_{-l})$

$$R(X) = \sum_{i=-l}^{k-1} x_i \cdot 2^i$$

**Useful Dependencies**

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

| $x_i$ | $1 - x_i$ | $\overline{x_i}$ | $|X|$ |
|------|-----------|-------------------|-----|
| 0    | 1         | 1                 | X when $X \geq 0$ |
| 1    | 0         | 0                 | - $X$ when $X < 0$ |
One's Complement Representation

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

\[ OC(A) \overset{\text{def}}{=} \overline{A} = 2^k - 2^{-l} - A \]

Properties:
\[ 0 \leq OC(A) \leq 2^k - 2^{-l} \]
\[ OC(OC(A)) = A \]
One's Complement Representation of Signed Numbers

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

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

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

One's Complement Mapping

\[
\begin{array}{c|c|c}
X > 0 & 0 & X < 0 \\
\hline
X & 0, 2^{k-1} & X + 2^{k-1} = 2^{k-1} - |X| \\
\end{array}
\]

\(k = 4\)
4-bit One's Complement Representation

Two's Complement Representation
Two's Complement Transformation (1)

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

\[ TC(A) = \begin{cases} \overline{A} + 2^{-l} = 2^k - A & \text{for } A > 0 \\ 0 & \text{for } A = 0 \end{cases} \]

Properties:
\[ 0 \leq TC(A) \leq 2^k - 2^{-l} \]
\[ TC(TC(A)) = A \]

Two's Complement Transformation (2)

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

\[ TC(A) = \overline{A} + 2^{-l} \mod 2^k = 2^k - A \mod 2^k \]
Two's Complement Representation of Signed Numbers

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

\[
R(X) = \begin{cases} 
X & \text{for } X \geq 0 \\
\text{TC}(|X|) & \text{for } X < 0 
\end{cases}
\]

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

Two's Complement Representation of Signed Integers

<table>
<thead>
<tr>
<th>$X&gt;0$</th>
<th>0</th>
<th>$X&lt;0$</th>
<th>$k=4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>0</td>
<td>$X+2^k = 2^k -</td>
<td>X</td>
</tr>
</tbody>
</table>

-8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7

-8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
One's Complement versus Two's Complement

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

- 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 $\rightarrow$ when dealing with positive and negative numbers
  - Unsigned binary $\rightarrow$ when dealing with non-negative numbers only
### Arithmetic Operations in Signed Number Representations

#### Unsigned Addition vs. Signed Addition

<table>
<thead>
<tr>
<th>Machine</th>
<th>Programmer</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight</td>
<td>128 64 32 16 8 4 2 1</td>
</tr>
<tr>
<td>carry</td>
<td>1 1 1</td>
</tr>
<tr>
<td>X</td>
<td>0 0 0 1 0 0 1 1</td>
</tr>
<tr>
<td>Y</td>
<td>1 0 0 0 0 1 0 1</td>
</tr>
<tr>
<td>= S</td>
<td>1 0 0 1 1 0 0 0</td>
</tr>
</tbody>
</table>

Diagram showing the addition process for both unsigned and signed numbers, with FA (Full Adder) logic gates and carry and sum outputs indicated.
### Out of range flags

**Carry flag - C**

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**

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)

---

### Overflow for Signed Numbers

**Indication of overflow**

Positive + Positive = Negative

Negative + Negative = Positive

**Formulas**

Overflow_2's complement = \overline{x_{k-1}} \overline{y_{k-1}} s_{k-1} + x_{k-1} y_{k-1} s_{k-1} =

= c_k \oplus c_{k-1}
4-bit Two's Complement Representation

Addition and Subtraction

Two’s complement

Numbers of the same sign

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-15</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-15</td>
</tr>
</tbody>
</table>

carry but not overflow

Numbers of the opposite sign

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-5</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-15</td>
</tr>
</tbody>
</table>

no carry but overflow

carry but not overflow
Subtraction

- To subtract $A - B$
  - 1) invert bits of $B$
  - 2) add ulp to $B$
  - 3) perform addition as on previous page: $A + (-B)$

- Example: $0011 - 0001$
  $0011 +$
  $1110 +$
  $1$
  $----$
  $1\overline{0}10$

- The ulp (LSB) can be added into the carryin part of an adder, saving hardware and making a universal adder/subtractor

Two's Complement Adder/Subtractor Architecture

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

**Two’s complement**

Sh.L \[00101_2 = +5\] = 01010_2 = +10

Sh.L \[11011_2 = -5\] = 10110_2 = -10

Sh.L \[01010_2 = +10\] = 10100_2 = -12

```
\[\text{Shift left may cause overflow}\]
```

Sh.R \[00101_2 = +5\] = \[00101_2 = +2\] rem 1

Sh.R \[11011_2 = -5\] = \[11011_2 = -3\] rem 1

```
\[\text{Shift right requires sign extension}\]
```

---

**Addition and Subtraction**

**One’s complement**

Numbers of the same sign

\[\begin{array}{c}
-15 \ 8 \ 4 \ 2 \ 1 \\
1 \ 1 \ 0 \ 1 \ 0 \\
1 \ 0 \ 1 \ 0 \ 1 \\
\hline
1 \ 0 \ 1 \ 1 \ 1
\end{array}\]  
\[\begin{array}{c}
-5 \\
-10 \\
\hline
+ 1 \rightarrow 1\\
1 \ 0 \ 0 \ 0 \ 0
\end{array}\]  

```
end-around carry
```

Numbers of the opposite sign

\[\begin{array}{c}
-15 \ 8 \ 4 \ 2 \ 1 \\
0 \ 1 \ 0 \ 1 \ 0 \\
1 \ 1 \ 0 \ 1 \ 0 \\
\hline
1 \ 0 \ 0 \ 1 \ 0 \ 0 \\
\hline
+ 1 \rightarrow 1 \\
0 \ 0 \ 1 \ 0 \ 1
\end{array}\]  

```
Disadvantage: Need another adder after the addition is complete!
```
### Arithmetic Shift

#### One’s complement

- Sh.L \( \{00101_2 = +5\} = 01010_2 = +10 \)
- Sh.L \( \{11010_2 = -5\} = 10101_2 = -10 \)
- Sh.L \( \{01010_2 = +10\} = 10100_2 = -11 \)

Shift left may cause overflow

- Sh.R \( \{00101_2 = +5\} = 00010_2 = +2 \) rem 1
- Sh.R \( \{11011_2 = -5\} = 11101_2 = -2 \) rem -1

Shift right requires sign extension

### Addition and Subtraction

#### Signed-magnitude

<table>
<thead>
<tr>
<th>Numbers of the same sign</th>
<th>Numbers of the opposite sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign bit</td>
<td>magnitude</td>
</tr>
<tr>
<td>0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0</td>
<td>1 0 0 0 1</td>
</tr>
<tr>
<td>carry = overflow</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Signed-Magnitude Adder

Signed-magnitude adder/subtractor is significantly more complex than a simple adder.

Two's-complement adder/subtractor needs very little hardware other than a simple adder.

Arithmetic with Biased Adders

Addition/subtraction of biased numbers:
\[
\begin{align*}
x + y + \text{bias} &= (x + \text{bias}) + (y + \text{bias}) - \text{bias} \\
x - y + \text{bias} &= (x + \text{bias}) - (y + \text{bias}) + \text{bias}
\end{align*}
\]

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.
Signed-Number Representations

Summary

Fixed-Point Number Systems

Fixed Point Number system

- Positional
  - Fixed-radix
  - Mixed-radix

- Non-positional
  - Conventional (unsigned)
    - Binary
    - Decimal
    - Hexadecimal
  - Unconventional (signed)
    - Signed-digit
      - Non-redundant
      - Redundant
### K-bit Signed Binary Numbers

<table>
<thead>
<tr>
<th>Representation</th>
<th>Representation for $X&gt;0$</th>
<th>Representation for 0</th>
<th>Representation for $X&lt;0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed-magnitude</td>
<td>$X$</td>
<td>0, $2^{k-1}$</td>
<td>$2^{k-1}+</td>
</tr>
<tr>
<td>Biased</td>
<td>$X+B$</td>
<td>$B$</td>
<td>$X+B$</td>
</tr>
<tr>
<td>Complement</td>
<td>$X$</td>
<td>0, $M \ mod \ 2^k$</td>
<td>$M-</td>
</tr>
<tr>
<td>Two’s complement</td>
<td>$X$</td>
<td>0</td>
<td>$2^{k-1}</td>
</tr>
<tr>
<td>One’s complement</td>
<td>$X$</td>
<td>0, $2^{k-ulp}$</td>
<td>$2^{k-ulp-</td>
</tr>
</tbody>
</table>

### Value of the Number in Signed Representations

<table>
<thead>
<tr>
<th>Representation</th>
<th>Value of $(x_{k-1} x_{k-2} \ldots x_1 x_0 x_{-1} \ldots x_{-l})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signed-magnitude</td>
<td>$X = (-1)^{x_{k-1}} \sum_{i=-l}^{k-2} x_i \cdot 2^i$</td>
</tr>
<tr>
<td>Biased</td>
<td>$X = \sum_{i=-l}^{k-1} x_i \cdot 2^i - B$</td>
</tr>
<tr>
<td>Two’s complement</td>
<td>$X = -x_{k-1} 2^{k-1} + \sum_{i=-l}^{k-2} x_i \cdot 2^i$</td>
</tr>
<tr>
<td>One’s complement</td>
<td>$X = -x_{k-1} (2^{k-1} - ulp) + \sum_{i=-l}^{k-2} x_i \cdot 2^i$</td>
</tr>
</tbody>
</table>
Extending Numbers of Bits of a Signed Number

\[
X = x_{k-1} x_{k-2} \ldots x_1 x_0 \cdot x_1 x_2 \ldots x_l
\]

\[
Y = y_{k-1} y_{k-2} \ldots y_k y_1 \cdot y_0 \cdot y_2 \ldots y_l y_{(l+1)} \ldots y_{l'}
\]

- **signed-magnitude**
  \[
x_{k-1} 0 0 0 0 0 0 \cdot x_2 \ldots x_1 x_0 \cdot x_1 x_2 \ldots x_l \ 0 0 0 0 0 0
\]

- **two's complement**
  \[
x_{k-1} x_{k-1} x_{k-1} \ldots x_{k-1} x_2 \ldots x_1 x_0 \cdot x_1 x_2 \ldots x_l \ 0 0 0 0 0 0
\]

- **one's complement**
  \[
x_{k-1} x_{k-1} x_{k-1} \ldots x_{k-1} x_2 \ldots x_1 x_0 \cdot x_1 x_2 \ldots x_l x_{k-1} \ldots x_{k-1}
\]

- **biased**
  \[
  \overline{x_{k-1}} \overline{x_{k-1}} \ldots \overline{x_{k-1}} x_{k-2} \ldots x_1 x_0 \cdot x_1 x_2 \ldots x_l \ 0 0 0 0 0 0
\]

Signed Numbers: Usability

- The vast majority of digital systems use either two's complement or unsigned binary
  - Two's complement \( \rightarrow \) when dealing with positive and negative numbers
    - VHDL: signed data type
      - 0000 \( \rightarrow \) 0
      - 0111 \( \rightarrow \) 7
      - 1000 \( \rightarrow \) -8
      - 1111 \( \rightarrow \) -1
  - Unsigned binary \( \rightarrow \) when dealing with non-negative numbers only
    - VHDL: unsigned data type
      - 0000 \( \rightarrow \) 0
      - 0111 \( \rightarrow \) 7
      - 1000 \( \rightarrow \) 8
      - 1111 \( \rightarrow \) 15
Generalized Complement Representation of Signed Integers
## Generalized Complement Representation of Signed Integers

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

<table>
<thead>
<tr>
<th>Desired operation</th>
<th>Computation to be performed mod $M$</th>
<th>Correct result with no overflow</th>
<th>Overflow condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ (+x) + (+y) $</td>
<td>$ x + y $</td>
<td>$ x + y $</td>
<td>$ x + y &gt; P $</td>
</tr>
<tr>
<td>$ (+x) + (-y) $</td>
<td>$ x + (M - y) $</td>
<td>$ x - y $ if $ y \leq x $</td>
<td>$ N/A $</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$ M - (y - x) $ if $ y &gt; x $</td>
<td></td>
</tr>
<tr>
<td>$ (-x) + (+y) $</td>
<td>$(M - x) + y $</td>
<td>$ y - x $ if $ x \leq y $</td>
<td>$ N/A $</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$ M - (x - y) $ if $ x &gt; y $</td>
<td></td>
</tr>
<tr>
<td>$ (-x) + (-y) $</td>
<td>$(M - x) + (M - y) $</td>
<td>$ M - (x + y) $</td>
<td>$ x + y &gt; N $</td>
</tr>
</tbody>
</table>