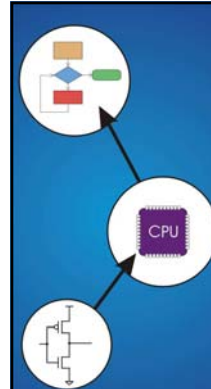




Introduction to Computer Engineering

ECE/CS 252, Fall 2010
Prof. Mikko Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin – Madison



Chapter 2 Bits, Data Types, and Operations - Part 2

Slides based on set prepared by
Gregory T. Byrd, North Carolina State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Review: Unsigned Integers

Weighted positional notation

- like decimal numbers: "329"
- "3" is worth 300, because of its position, while "9" is only worth 9

$$\begin{array}{ccc} & 3 & 2 & 9 \\ & 10^2 & 10^1 & 10^0 \\ \hline & 3 \times 100 & + 2 \times 10 & + 9 \times 1 = 329 \end{array}$$
$$\begin{array}{ccc} & 1 & 0 & 1 \\ & 2^2 & 2^1 & 2^0 \\ \hline & 1 \times 4 & + 0 \times 2 & + 1 \times 1 = 5 \end{array}$$

2-3

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Review: Unsigned Integers (cont.)

An n -bit unsigned integer represents 2^n values:
from 0 to $2^n - 1$.

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

2-4

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Review: Unsigned Binary Arithmetic

Base-2 addition – just like base-10!

- add from right to left, propagating carry

$$\begin{array}{r} 10010 \\ + 1001 \\ \hline 11011 \end{array}$$
$$\begin{array}{r} 10010 \\ + 1011 \\ \hline 11101 \end{array}$$
$$\begin{array}{r} 1111 \\ + 1 \\ \hline 10000 \end{array}$$
$$\begin{array}{r} 10111 \\ + 111 \\ \hline 11110 \end{array}$$

2-5

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

New: Signed Integers

With n bits, we have 2^n distinct values.

- assign about half to positive integers (1 through 2^{n-1}) and about half to negative (-2^{n-1} through -1)
- that leaves two values: one for 0, and one extra

Positive integers

- just like unsigned – zero in most significant bit
 $00101 = 5$

Negative integers

- sign-magnitude – set top bit to show negative, other bits are the same as unsigned
 $10101 = -5$
- one's complement – flip every bit to represent negative
 $11010 = -5$
- in either case, MS bit indicates sign: 0=positive, 1=negative

2-6

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Two's Complement

Problems with sign-magnitude and 1's complement

- two representations of zero (+0 and -0)
- arithmetic circuits are complex
 - Adding a negative number => subtraction
 - Need to "correct" result to account for borrowing

Two's complement representation developed to make circuits easy for arithmetic.

- for each positive number (X), assign value to its negative (-X), such that $X + (-X) = 0$ with "normal" addition, ignoring carry out

00101 (5)	01001 (9)
+ 11011 (-5)	+ 10111 (-9)
00000 (0)	00000 (0)

2-7

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Two's Complement Representation

If number is positive or zero,

- normal binary representation, zeroes in upper bit(s)

If number is negative,

- start with positive number
- flip every bit (i.e., take the one's complement)
- then add one

00101 (5) 11010 (1's comp) + 1 11011 (-5)	01001 (9) 10110 (1's comp) + 1 10111 (-9)
--	--

2-8

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Two's Complement Shortcut

To take the two's complement of a number:

- copy bits from right to left until (and including) the first "1"
- flip remaining bits to the left

011010000 100101111 (1's comp) + 1 100110000	011010000 (flip) ↓ (copy) ↓ 100110000
---	---

2-9

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Two's Complement Signed Integers

MS bit is sign bit – it has weight -2^{n-1} .

Range of an n-bit number: -2^{n-1} through $2^{n-1} - 1$.

- The most negative number (-2^{n-1}) has no positive counterpart.

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

2-10

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Converting 2's Complement to Decimal

1. If leading bit is one, take two's complement to get a positive number.
2. Add powers of 2 that have "1" in the corresponding bit positions.
3. If original number was negative, add a minus sign.

$X = 11100110_{\text{two}}$ $-X = 00011010$ $= 2^4 + 2^3 + 2^1 = 16 + 8 + 2$ $= 26_{\text{ten}}$ $X = -26_{\text{ten}}$	<table> <tr><th>n</th><th>2^n</th></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>2</td></tr> <tr><td>2</td><td>4</td></tr> <tr><td>3</td><td>8</td></tr> <tr><td>4</td><td>16</td></tr> <tr><td>5</td><td>32</td></tr> <tr><td>6</td><td>64</td></tr> <tr><td>7</td><td>128</td></tr> <tr><td>8</td><td>256</td></tr> <tr><td>9</td><td>512</td></tr> <tr><td>10</td><td>1024</td></tr> </table>	n	2^n	0	1	1	2	2	4	3	8	4	16	5	32	6	64	7	128	8	256	9	512	10	1024
n	2^n																								
0	1																								
1	2																								
2	4																								
3	8																								
4	16																								
5	32																								
6	64																								
7	128																								
8	256																								
9	512																								
10	1024																								

Assuming 8-bit 2's complement numbers.

2-11

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Converting Decimal to Binary (2's C)

1. Change to positive decimal number.
2. Use either *repeated division by 2* or *repeated subtraction of powers of two*
3. Append a zero as MS bit; if original was negative, take two's complement.

$X = -104_{\text{ten}}$ $104 - 64 = 40$ bit 6 $40 - 32 = 8$ bit 5 $8 - 8 = 0$ bit 3 01101000_{two} $X = 10011000_{\text{two}}$	<table> <tr><th>n</th><th>2^n</th></tr> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>2</td></tr> <tr><td>2</td><td>4</td></tr> <tr><td>3</td><td>8</td></tr> <tr><td>4</td><td>16</td></tr> <tr><td>5</td><td>32</td></tr> <tr><td>6</td><td>64</td></tr> <tr><td>7</td><td>128</td></tr> <tr><td>8</td><td>256</td></tr> <tr><td>9</td><td>512</td></tr> <tr><td>10</td><td>1024</td></tr> </table>	n	2^n	0	1	1	2	2	4	3	8	4	16	5	32	6	64	7	128	8	256	9	512	10	1024
n	2^n																								
0	1																								
1	2																								
2	4																								
3	8																								
4	16																								
5	32																								
6	64																								
7	128																								
8	256																								
9	512																								
10	1024																								

2-12

Operations: Arithmetic and Logical

Recall:

a data type includes *representation* and *operations*.

We now have a good representation for signed integers, so let's look at some arithmetic operations:

- Addition
- Subtraction
- Sign Extension

(We'll also look at overflow conditions for addition.)

Multiplication, division, etc., can be built from these basic operations.

Review: Logical operations are also useful:

- AND
- OR
- NOT

2-13

Addition

As we've discussed, 2's comp. addition is just binary addition.

- assume all integers have the same number of bits
- ignore carry out
- for now, assume that sum fits in n-bit 2's comp. representation

$$\begin{array}{r} 01101000 \text{ (104)} \\ + \underline{11110000 \text{ (-16)}} \\ 01011000 \text{ (98)} \end{array} \quad \begin{array}{r} 11110110 \text{ (-10)} \\ + \underline{11110111 \text{ (-9)}} \\ 11101101 \text{ (-19)} \end{array}$$

Assuming 8-bit 2's complement numbers.

2-14

Subtraction

Negate subtrahend (2nd no.) and add.

- assume all integers have the same number of bits
- ignore carry out
- for now, assume that difference fits in n-bit 2's comp. representation

$$\begin{array}{r} 01101000 \text{ (104)} \\ - \underline{00010000 \text{ (16)}} \\ 01101000 \text{ (104)} \\ + \underline{\hspace{1cm} \text{ (-16)}} \\ \hspace{1cm} \text{ (88)} \end{array}$$

Assuming 8-bit 2's complement numbers.

2-15

Sign Extension

To add two numbers, we must represent them with the same number of bits.

If we just pad with zeroes on the left:

$$\begin{array}{r} \text{4-bit} \quad \quad \quad \text{8-bit} \\ 0100 \text{ (4)} \quad \quad 00000100 \text{ (still 4)} \\ 1100 \text{ (-4)} \quad \quad 00001100 \text{ (12, not -4)} \end{array}$$

Instead, replicate the MS bit -- the sign bit:

$$\begin{array}{r} \text{4-bit} \quad \quad \quad \text{8-bit} \\ 0100 \text{ (4)} \quad \quad 00000100 \text{ (still 4)} \\ 1100 \text{ (-4)} \quad \quad 11111100 \text{ (still -4)} \end{array}$$

2-16

Overflow

If operands are too big, then sum cannot be represented as an n-bit 2's comp number.

$$\begin{array}{r} \boxed{01000} \text{ (8)} \quad \quad \boxed{11000} \text{ (-8)} \\ + \underline{\boxed{01001} \text{ (9)}} \quad \quad + \underline{\boxed{10111} \text{ (-9)}} \\ \boxed{10001} \text{ (-15)} \quad \quad \boxed{01111} \text{ (+15)} \end{array}$$

We have overflow if:

- signs of both operands are the same, and
- sign of sum is different.

Another test -- easy for hardware:

- carry into MS bit does not equal carry out

2-17

Fractions: Fixed-Point

How can we represent fractions?

- Use a "binary point" to separate positive from negative powers of two -- just like "decimal point."
- 2's comp addition and subtraction still work.
 - if binary points are aligned

$$\begin{array}{r} \begin{array}{l} \swarrow 2^{-1} = 0.5 \\ \swarrow 2^{-2} = 0.25 \\ \swarrow 2^{-3} = 0.125 \end{array} \\ 00101000.101 \text{ (40.625)} \\ + \underline{11111110.110 \text{ (-1.25)}} \\ 00100111.011 \text{ (39.375)} \end{array}$$

No new operations -- same as integer arithmetic.

2-18

Very Large and Very Small: Floating-Point

Large values: 6.023×10^{23} -- requires 79 bits

Small values: 6.626×10^{-34} -- requires >110 bits

Use equivalent of "scientific notation": $F \times 2^E$

Need to represent F (*fraction*), E (*exponent*), and sign.

IEEE 754 Floating-Point Standard (32-bits):



$$N = -1^S \times 1.\text{fraction} \times 2^{\text{exponent}-127}, \quad 1 \leq \text{exponent} \leq 254$$

$$N = -1^S \times 0.\text{fraction} \times 2^{-126}, \quad \text{exponent} = 0$$

2-19

Floating Point Example

Single-precision IEEE floating point number:



- Sign is 1 -- number is negative.
- Exponent field is 01111110 = 126 (decimal).
- Fraction is 0.100000000000... = 0.5 (decimal).

$$\text{Value} = -1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = \mathbf{-0.75}.$$

2-20

LC-3 Data Types

Some data types are supported directly by the instruction set architecture.

For LC-3, there is only one supported data type:

- 16-bit 2's complement signed integer
- Operations: ADD, AND, NOT

Other data types are supported by interpreting 16-bit values as logical, text, fixed-point, etc., in the software that we write.

2-21

Summary

Review: unsigned numbers

New: signed numbers

Sign/magnitude and one's complement

Two's complement

Two's Complement operations & issues

Addition, subtraction

Sign extension

Overflow

Fractions

Fixed point

Floating point: IEEE754 standard

2-22