CPT_S 260 Intro to Computer Architecture
Lecture 5

Integer Arithmetic
January 24, 2022

Ganapati Bhat
School of Electrical Engineering and Computer Science
Washington State University

Recap: Numbering Systems

- **A number system of a specific base (radix) uses numbers from 0 to that base-1**
- **Numbers can be computed to decimal through the sum of the weighted digits-**

$$\boldsymbol{Number} = \sum_{i=0}^{n} base^i * digit$$

	Binary	Octal	Decimal	Hexadecimal
Base	2	8	10	16
Symbols	{0,1}	{0,1,2,3,4,5,6,7}	{0,1,2,3,4,5,6,7,8,9}	{0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F}

Recap: Decimal → Binary

- **Choose one of two equivalent methods**
  – Subtract largest power of 2 until difference = 0. For each subtraction, place ‘1’ in a binary string at a position corresponding to the power. Fill empty positions with 0’s.
  – Divide by 2 until quotient = 0. For each division, place the remainder, either ‘0’ or ‘1’, in a binary string growing from right to left. The rightmost end of the string is the low-order bit.

<u>Example</u>: Convert 131_{10} to binary

Subtract largest power

$131 - 2^7 = 3$
$3 - 2^1 = 1$
$1 - 2^0 = 0$

$0*2^6$
$0*2^5$
$0*2^4$
$0*2^3$
$0*2^2$

Divide by 2

131 / 2 = 65 R	1
65 / 2 = 32 R	1
32 / 2 = 16 R	0
16 / 2 = 8 R	0
8 / 2 = 4 R	0
4 / 2 = 2 R	0
2 / 2 = 1 R	0
1 / 2 = 0 R	1

$(10000011)_2$

Example

Recap: Hexadecimal

Decimal	Hex
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Hexadecimal represents numbers as a series of base-16 expressions multiplied by integers from zero to 15.

<u>Example:</u> 47_{10} represented in hex

$$(47)_{10}$$

$$32 + 15$$

$$2*16^1 + 15*16^0$$

$$(2F)_{16}$$

Binary ↔ Hexadecimal

Partition binary digits right-to-left in groups of four.
Convert each group to one hexadecimal symbol.

Example: Convert 1000101010_2 to hexadecimal

(0010	0010	$1010)_2$
2	2	A

$$\mathbf{2*16^2 + 2*16^1 + A*16^0}$$

$$\mathbf{(22A)_{16}}$$

Hexadecimal number	Binary-coded hexadecimal
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Hexadecimal → Binary

Convert each hexadecimal symbol to a four-digit binary number and combine.

<u>Example</u>: Convert $F01_{16}$ to binary

(F	0	$1)_{16}$
1111	0000	0001

$(111100000001)_2$

Hexadecimal number	Binary-coded hexadecimal
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Decimal to Hexadecimal

- Divide by 16 until quotient = 0. For each division, place the remainder, in the hexadecimal string growing from right to left. The rightmost end of the string is the low-order bit.

Decimal	Hex
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Sign in Binary

- **Represent the domain of negative and positive integers in one of two distinct ways**
- **Signed Magnitude**
  – In signed-magnitude format, one bit designates sign and the remaining bits magnitude. The sign bit is the high-order bit at the leftmost position and takes ‘1’ for negative or ‘0’ for positive. The magnitude is an absolute value.
- **2’s complement**
  – In signed-2’s-complement format, positive integers take the same form as in signed-magnitude but negative integers do not. The 2’s-complement is one plus the 1’s-complement, which, for a given number and range, is the difference between the range’s maximum value and the number.

2's-Complement Signed Integers

- **Given an n-bit number:**

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- **Range: -2^{n-1} to $+2^{n-1} - 1$**
- **Example**
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^{2} + 0\times2^{1} + 0\times2^{0}$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- **Using 32 bits, 2's complement range is:**
  - –2,147,483,648 to +2,147,483,647

2's Complement Range for 32 bits

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
$\ldots \qquad \ldots$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2{,}147{,}483{,}648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -2{,}147{,}483{,}646_{ten}$
$\ldots \qquad \ldots$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1_{ten}$

Signed Negation in 2's Complement

- **Complement and add 1**
  - Complement means 1 → 0, 0 → 1

- **Example: negate +2**
  - **+2 = 0000 0000 … 0010_2**
  - **–2 = 1111 1111 … 1101_2 + 1**
    = 1111 1111 … 1110_2

- **How does this work?**
- **A number and it's complement add up to all 1's**
  - $(11111111 \ldots 11)_2$
- **We know $(11111111 \ldots 11)_2 = -1$**
- **That is,**

$$x + \bar{x} = -1$$
$$\bar{x} = x + 1$$

Signed-2's-complement

<u>Example:</u> $(260)_{10}$ represented in 16-bit signed-2's-complement format

$(0000\ 0001\ 0000\ 0100)_2$

<u>Example:</u> $(-260)_{10}$ represented in 16-bit signed-2's-complement format

$(0000\ 0001\ 0000\ 0100)_2$

Flip bits

(1	111	1110	1111	$1011)_2$
-2^{15}	2^{14} 2^{13} 2^{12}	2^{11} 2^{10} 2^{9} 2^{8}	2^{7} 2^{6} 2^{5} 2^{4}	2^{3} 2^{2} 2^{1} 2^{0})

Add one

(1	111	1110	1111	$1100)_2$
-2^{15}	2^{14} 2^{13} 2^{12}	2^{11} 2^{10} 2^{9} 2^{8}	2^{7} 2^{6} 2^{5} 2^{4}	2^{3} 2^{2} 2^{1} 2^{0})

Sign Extension

- Convert a binary number represented in N bits to a number represented with more than N bits
- For example, the computer has 32-bit memory, and the number is only 16 bits
- Sign extension allows us to convert
- Fill the lower order bits with the original number
- Copy the sign bit of the number to the left and fill the bits

Computer Arithmetic

- Computers store and perform operations on information encoded in bits, which are binary digits of value 0 or 1.
- For addition on integers, computers add bits of operands starting at the low-order bit.
- When the sum of operand bits is greater than 1, the sum carries to the higher bit.
- Likewise, subtraction occurs by negating the appropriate operands and performing addition.

Addition

Example: $(360)_{10} + (260)_{10}$ in 16-bit unsigned format

Example: $(-360)_{10} + (-260)_{10}$ in 16-bit signed-2's-complement format

Addition

Example: $(360)_{10} + (260)_{10}$ in 16-bit unsigned format

$$
\begin{array}{rl}
 & 1 \\
 & (0000\ 0001\ 0110\ 1000)_2 \\
+ & \underline{(0000\ 0001\ 0000\ 0100)}_2 \\
 & (0000\ 0010\ 0110\ 1100)_2
\end{array}
$$

Example: $(-360)_{10} + (-260)_{10}$ in 16-bit signed-2's-complement format

$(0000\ 0001\ 0110\ 1000)_2$

Flip bits

$(1111\ 1110\ 1001\ 0111)_2$

Add one

$(1111\ 1110\ 1001\ 1000)_2$

$(0000\ 0001\ 0000\ 0100)_2$

Flip bits

$(1111\ 1110\ 1111\ 1011)_2$

Add one

$(1111\ 1110\ 1111\ 1100)_2$

$$
\begin{array}{rl}
 & 1111\ 11\ \ 1\ 1111 \\
 & (1111\ 1110\ 1001\ 1000)_2 \\
+ & \underline{(1111\ 1110\ 1111\ 1100)}_2 \\
 & (1111\ 1101\ 1001\ \ 0100)_2
\end{array}
$$

Subtraction

- **Example:** $(360)_{10}$ - $(260)_{10}$ in 16-bit unsigned format

- **Example:** $(-360)_{10}$ - $(-260)_{10}$ in 16-bit signed-2's-complement format

Subtraction

<u>Example:</u> $(360)_{10}$ - $(260)_{10}$ in 16-bit unsigned format

$$
\begin{array}{rl}
 & {}^{2} \\
 & (0000\ 0001\ 0110\ \cancel{1}000)_2 \\
- & \underline{(0000\ 0001\ 0000\ 0100)}_2 \\
 & (0000\ 0000\ 0110\ 0100)_2
\end{array}
$$

<u>Example:</u> $(-360)_{10}$ - $(-260)_{10}$ in 16-bit signed-2's-complement format

$(0000\ 0001\ 0110\ 1000)_2$

Flip bits

$(1111\ 1110\ 1001\ 0111)_2$

Add one

$(1111\ 1110\ 1001\ 1000)_2$

$(0000\ 0001\ 0000\ 0100)_2$

$$
\begin{array}{rl}
 & (1111\ 1110\ 1001\ 1000)_2 \\
+ & \underline{(0000\ 0001\ 0000\ 0100)}_2 \\
 & (1111\ 1111\ 1001\ 1100)
\end{array}
$$

Overflow

- The addition of two numbers with the same sign or subtraction of two numbers with different sign may cause overflow

- The condition wherein a result cannot be represented in allocated memory

- An overflow condition can be detected by observing the carry into the sign bit and carry out of the sign bit

carries:	0 1		carries:	1 0	
+70	0	1000110	−70	1	0111010
+80	0	1010000	−80	1	0110000
+150	1	0010110	−150	0	1101010

Conditions for Overflow

Operation	A	B	C
A + B = C	> 0	> 0	if (C <= 0)
	> 0	< 0	no overflow
	< 0	> 0	no overflow
	< 0	< 0	if (C >= 0)
A - B = C	> 0	> 0	no overflow
	> 0	< 0	if (C <= 0)
	< 0	> 0	if (C >= 0)
	< 0	< 0	no overflow

Overflow Detection in Binary

- **We need to compare three bits to check for overflow**
- **Needs a three-bit comparator**
- **Efficient method to check overflow**
- **Compare the carry in and carry out for the MSB**
  - If they are different then there is an overflow
  - If not, there is no overflow
- **Example**

$$
\begin{array}{rl}
\text{carries: } 0 & 1 \\
+70 & 0\ 1000110 \\
+\underline{80} & \underline{0\ 1010000} \\
+150 & 1\ 0010110
\end{array}
\qquad\qquad
\begin{array}{rl}
\text{carries: } 1 & 0 \\
-70 & 1\ 0111010 \\
-\underline{80} & \underline{1\ 0110000} \\
-150 & 0\ 1101010
\end{array}
$$