CMSC 313 Lecture 02

- Bits of Memory
- Data formats for negative numbers
  - ◇ signed magnitude
  - ◇ one's complement
  - ◇ two's complement
  - ◇ excess bias
- Modulo arithmetic & two's complement

Base Conversion with the Remainder Method

- **<u>Example:</u> Convert 23.375_{10} to base 2. Start by converting the integer portion:**

		Integer	Remainder	
23/2	=	11	1	Least significant bit
11/2	=	5	1	
5/2	=	2	1	
2/2	=	1	0	
1/2	=	0	1	Most significant bit

$$(23)_{10} = (10111)_2$$

Base Conversion with the Multiplication Method

- **Now, convert the fraction:**

Most significant bit

$$.375 \times 2 = 0.75$$

$$.75 \times 2 = 1.5$$

$$.5 \times 2 = 1.0$$

Least significant bit

$$(.375)_{10} = (.011)_2$$

- **Putting it all together, $23.375_{10} = 10111.011_2$.**

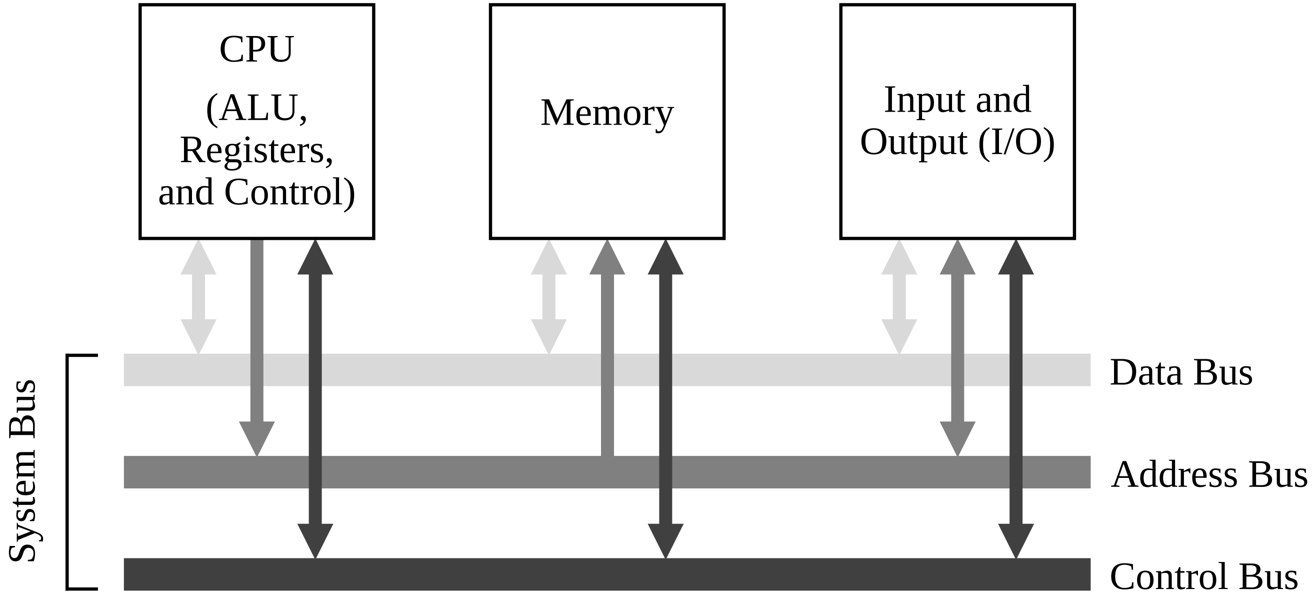
Nonterminating Base 2 Fraction

- **We can't always convert a terminating base 10 fraction into an equivalent terminating base 2 fraction:**

$$.2 \times 2 = 0.4$$
$$.4 \times 2 = 0.8$$
$$.8 \times 2 = 1.6$$
$$.6 \times 2 = 1.2$$
$$.2 \times 2 = 0.4$$
$$\vdots$$

The System Bus Model

- A refinement of the von Neumann model, the system bus model has a CPU (ALU and control), memory, and an input/output unit.
- Communication among components is handled by a shared pathway called the *system bus*, which is made up of the data bus, the address bus, and the control bus. There is also a power bus, and some architectures may also have a separate I/O bus.



Random Access Memory (RAM)

- A single byte of memory holds 8 binary digits (bits).
- Each byte of memory has its own address.
- A 32-bit CPU can address 4 gigabytes of memory, but a machine may have much less (e.g., 256MB).
- For now, think of RAM as one big array of bytes.
- The data stored in a byte of memory is not typed.
- The assembly language programmer must remember whether the data stored in a byte is a character, an unsigned number, a signed number, part of a multi-byte number, ...

Signed Fixed Point Numbers

- For an 8-bit number, there are $2^8 = 256$ possible bit patterns. These bit patterns can represent negative numbers if we choose to assign bit patterns to numbers in this way. We can assign half of the bit patterns to negative numbers and half of the bit patterns to positive numbers.
- Four signed representations we will cover are:

  Signed Magnitude

  One's Complement

  Two's Complement

  Excess (Biased)

Signed Magnitude

- Also know as “sign and magnitude,” the leftmost bit is the sign (0 = positive, 1 = negative) and the remaining bits are the magnitude.

- <u>Example:</u>

  $+25_{10} = 00011001_2$

  $-25_{10} = 10011001_2$

- Two representations for zero: $+0 = 00000000_2$, $-0 = 10000000_2$.
- Largest number is +127, smallest number is -127_{10}, using an 8-bit representation.

One’s Complement

- **The leftmost bit is the sign (0 = positive, 1 = negative). Negative of a number is obtained by subtracting each bit from 2 (essentially, *complementing* each bit from 0 to 1 or from 1 to 0). This goes both ways: converting positive numbers to negative numbers, and converting negative numbers to positive numbers.**

- **<u>Example:</u>**

  $+25_{10} = 00011001_2$

  $-25_{10} = 11100110_2$

- **Two representations for zero: $+0 = 00000000_2$, $-0 = 11111111_2$.**
- **Largest number is $+127_{10}$, smallest number is -127_{10}, using an 8-bit representation.**

Two's Complement

- The leftmost bit is the sign (0 = positive, 1 = negative). Negative of a number is obtained by adding 1 to the one's complement negative. This goes both ways, converting between positive and negative numbers.

- <u>Example</u> (recall that -25_{10} in one's complement is 11100110_2):

  $+25_{10} = 00011001_2$

  $-25_{10} = 11100111_2$

- One representation for zero: $+0 = 00000000_2$, $-0 = 00000000_2$.
- Largest number is $+127_{10}$, smallest number is -128_{10}, using an 8-bit representation.

Excess (Biased)

- **The leftmost bit is the sign (usually 1 = positive, 0 = negative). Positive and negative representations of a number are obtained by adding a bias to the two's complement representation. This goes both ways, converting between positive and negative numbers. The effect is that numerically smaller numbers have smaller bit patterns, simplifying comparisons for floating point exponents.**

- **<u>Example</u> (excess 128 "adds" 128 to the two's complement version, ignoring any carry out of the most significant bit) :**

  $+12_{10} = 10001100_2$

  $-12_{10} = 01110100_2$

- **One representation for zero: $+0 = 10000000_2$, $-0 = 10000000_2$.**
- **Largest number is $+127_{10}$, smallest number is -128_{10}, using an 8-bit representation.**

Example: Convert -123

- **Signed Magnitude**

$123_{10} = 64 + 32 + 16 + 8 + 2 + 1 = 0111\ 1011_2$
$-123_{10} \Rightarrow 1111\ 1011_2$

- **One's Complement (flip the bits)**

$-123_{10} \Rightarrow 1000\ 0100_2$

- **Two's Complement (add 1 to one's complement)**

$-123_{10} \Rightarrow 1000\ 0101_2$

- **Excess 128 (add 128 to two's complement)**

$-123_{10} \Rightarrow 0000\ 0101_2$

3-bit Signed Integer Representations

Decimal	Unsigned	Sign Mag	1's Comp	2's Comp	Excess 4
7	111				
6	110				
5	101				
4	100				
3	011	011	011	011	111
2	010	010	010	010	110
1	001	001	001	001	101
0	000	000/100	000/111	000	100
-1		101	110	111	011
-2		110	101	110	010
-3		111	100	101	001
-4				100	000

Binary Addition

- **This simple binary addition example provides background for the signed number representations to follow.**

Carry in	0	0	0	0	1	1	1	1
Operands	0	0	1	1	0	0	1	1
	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

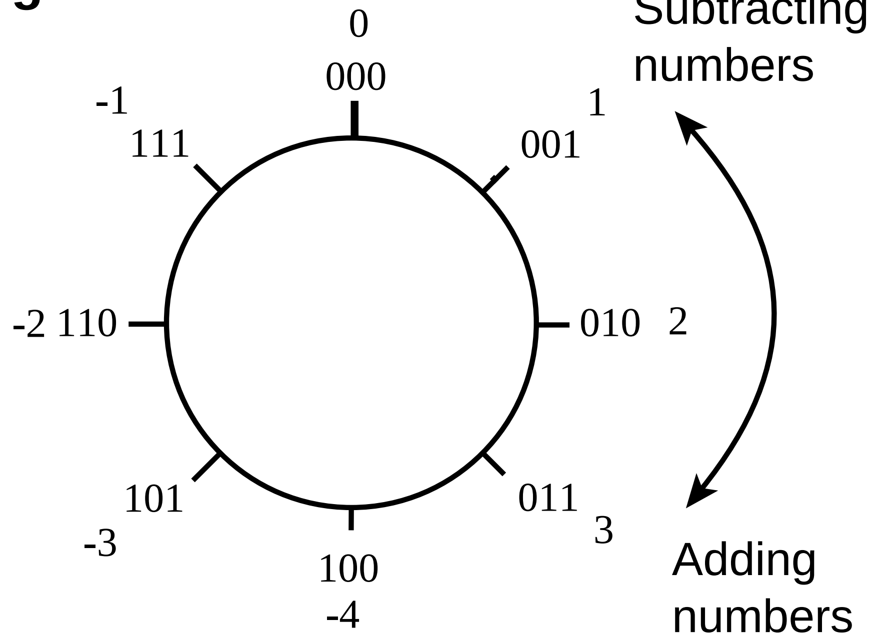
Carry out Sum

Example:

Carry		1	1	1	1	0	0	0	0	
Addend: *A*		0	1	1	1	1	1	0	0	$(124)_{10}$
Augend: *B*	+	0	1	0	1	1	0	1	0	$(90)_{10}$
Sum		1	1	0	1	0	1	1	0	$(214)_{10}$

Number Circle for 3-Bit Two's Complement Numbers

- Numbers can be added or subtracted by traversing the number circle clockwise for addition and counterclockwise for subtraction.
- Overflow occurs when a transition is made from +3 to -4 while proceeding around the number circle when adding, or from -4 to +3 while subtracting.



8-bit Two's Complement Addition

$$\begin{array}{rrcl} & 54_{10} & = & 0011\ 0110 \\ + & -48_{10} & = & 1101\ 0000 \\ \hline & 6_{10} & = & 0000\ 0110 \end{array}$$

$$\begin{array}{rrcl} & 44_{10} & = & 0010\ 1100 \\ + & -48_{10} & = & 1101\ 0000 \\ \hline & -4_{10} & = & 1111\ 1100 \end{array}$$

$$\begin{array}{rrcl} & -44_{10} & = & 1101\ 0100 \\ + & -48_{10} & = & 1101\ 0000 \\ \hline & -92_{10} & = & 1010\ 0100 \end{array}$$

One’s Complement Addition

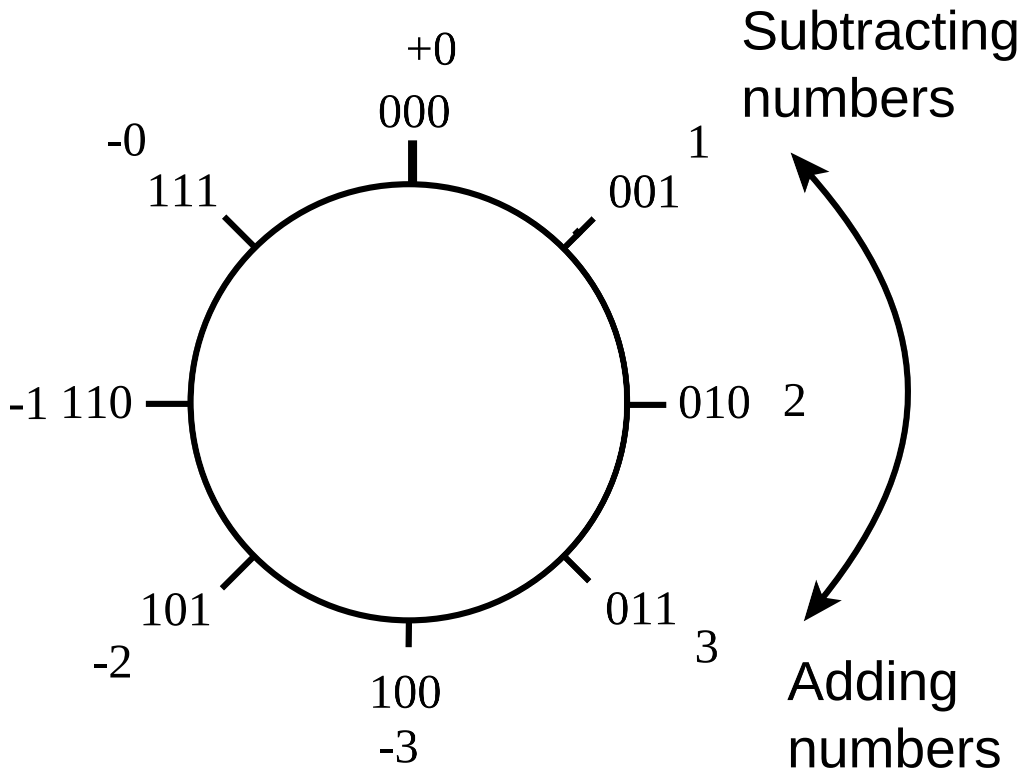
- **An example of one’s complement integer addition with an end-around carry:**

$$\begin{array}{rl} 1\ 0\ 0\ 1\ 1 & (-12)_{10} \\ +\ 0\ 1\ 1\ 0\ 1 & (+13)_{10} \\ \hline 1\ 0\ 0\ 0\ 0\ 0 & \\ & \text{End-around carry} \\ +\qquad\qquad 1 & \\ \hline 0\ 0\ 0\ 0\ 1 & (+1)_{10} \end{array}$$

- **The end-around carry is needed because there are two representations for 0 in one’s complement. Both representations for 0 are visited when one or both operands are negative.**

Number Circle (Revisited)

- **Number circle for a three-bit signed one's complement representation. Notice the two representations for 0.**



End-Around Carry for Fractions

- The end-around carry complicates one’s complement addition for non-integers, and is generally not used for this situation.
- The issue is that the distance between the two representations of 0 is 1.0, whereas the rightmost fraction position is less than 1.

$$
\begin{array}{r l}
0\,1\,0\,1\,.\,1 & (+5.5)_{10} \\
\cancel{+\quad 1\,1\,1\,0\,.\,0} & (-1.0)_{10} \\
\hline
1\,0\,0\,1\,1\,.\,1 & \\
+ \quad \rightarrow \cancel{1\,.\,0} & \\
\hline
0\,1\,0\,0\,.\,1 & (+4.5)_{10}
\end{array}
$$

End-Around Carry for Fractions

- The end-around carry complicates one's complement addition for non-integers, and is generally not used for this situation.
- The issue is that the distance between the two representations of 0 is 1.0, whereas the rightmost fraction position is less than 1.

$$
\begin{array}{rrl}
 & 0\,1\,0\,1\,.\,1 & (+5.5)_{10} \\
+ & 1\,1\,1\,0\,.\,1 & (-1.0)_{10} \\
\hline
 & 1\,0\,1\,0\,0\,.\,0 & \\
+ & 0\,.\,1 & \\
\hline
 & 0\,1\,0\,0\,.\,1 & (+4.5)_{10}
\end{array}
$$

Two's Complement Overflow

- An overflow occurs if adding two positive numbers yields a negative result or if adding two negative numbers yields a positive result.
- Adding a positive and a negative number never causes an overflow.
- Carry out of the most significant bit does not indicate an overflow.
- An overflow occurs when the carry into the most significant bit differs from the carry out of the most significant bit.

Two's Complement Overflow Examples

$$\begin{array}{rrcl} & 54_{10} & = & 0011\ 0110 \\ + & 108_{10} & = & 0110\ 1100 \\ \hline & 162_{10} & \neq & 1010\ 0010 \end{array}$$

$$\begin{array}{rrcl} & -103_{10} & = & 1001\ 1001 \\ + & -48_{10} & = & 1101\ 0000 \\ \hline & -151_{10} & \neq & 0110\ 1001 \end{array}$$

Is Two's Complement "Magic"?

- Why does adding positive and negative numbers work?
- Why do we add 1 to the one's complement to negate?

- Answer: Because modulo arithmetic works.

Modulo Arithmetic

- Definition: Let a and b be integers and let m be a positive integer. We say that $a \equiv b \pmod{m}$ if the remainder of a divided by m is equal to the remanider of b divided by m.

- In the C programming language, $a \equiv b \pmod{m}$ would be written

```
a % m == b % m
```

- We use the theorem:

  If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$
  then $a + c \equiv b + d \pmod{m}$.

A Theorem of Modulo Arithmetic

Thm: If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ then $a + c \equiv b + d \pmod{m}$.

Example: Let $m = 8$, $a = 3$, $b = 27$, $c = 2$ and $d = 18$.

$3 \equiv 27 \pmod{8}$ and $2 \equiv 18 \pmod{8}$.

$5 \equiv 45 \pmod{8}$.

Proof: Write $a = q_a m + r_a$, $b = q_b m + r_b$, $c = q_c m + r_c$ and $d = q_d m + r_d$, where r_a, r_b, r_c and r_d are between 0 and $m - 1$. Then,

$$a + c = (q_a + q_c)m + r_a + r_c$$

$$b + d = (q_b + q_d)m + r_b + r_d = (q_b + q_d)m + r_a + r_c.$$

Thus, $a + c \equiv r_a + r_c \equiv b + d \pmod{m}$.

Consider Numbers Modulo 256

$$
\begin{array}{rcrcrcrcr}
0000\ 0000_2 & = & 0 & \equiv & -256 & \equiv & 256 & \equiv & 512 \\
0000\ 0001_2 & = & 1 & \equiv & -255 & \equiv & 257 & \equiv & 513 \\
0000\ 0010_2 & = & 2 & \equiv & -254 & \equiv & 258 & \equiv & 514 \\
\vdots & & & & & & & & \\
0000\ 1111_2 & = & 15 & \equiv & -241 & \equiv & 271 & \equiv & 527 \\
\vdots & & & & & & & & \\
0111\ 1111_2 & = & 127 & \equiv & -129 & \equiv & 383 & \equiv & 639 \\
1000\ 0000_2 & = & 128 & \equiv & -128 & \equiv & 384 & \equiv & 640 \\
\vdots & & & & & & & & \\
1000\ 1111_2 & = & 143 & \equiv & -113 & \equiv & 399 & \equiv & 655 \\
\vdots & & & & & & & & \\
1111\ 0011_2 & = & 243 & \equiv & -13 & \equiv & 499 & \equiv & 755 \\
\vdots & & & & & & & & \\
1111\ 1111_2 & = & 255 & \equiv & -1 & \equiv & 511 & \equiv & 767
\end{array}
$$

If $0000\ 0000_2$ thru $0111\ 1111_2$ represents 0 thru 127 and $1000\ 0000_2$ thru $1111\ 1111_2$ represents -128 thru -1, then the most significant bit can be used to determine the sign.

Some Answers

- In 8-bit two's complement, we use addition modulo $2^8 = 256$, so adding 256 or subtracting 256 is equivalent to adding 0 or subtracting 0.

- To negate a number x, $0 \leq x \leq 128$:

$$-x = 0 - x \equiv 256 - x = (255 - x) + 1 = (\texttt{1111 1111}_2 - x) + 1$$

  Note that $\texttt{1111 1111}_2 - x$ is the one's complement of x.

- Now we can just add positive and negative numbers. For example:

$$3 + (-5) \equiv 3 + (256 - 5) = 3 + 251 = 254 \equiv 254 - 256 = -2.$$

  or two negative numbers (as long as there's no overflow):

$$(-3) + (-5) \equiv (256 - 3) + (256 - 5) = 504 \equiv 504 - 512 = -8.$$

For the following questions, *show all of your work*. It is not sufficient to provide the answers.

Exercise 1. Convert the following numbers.

a. 62347_{10} to unsigned binary

b. $8DF6_{16}$ to base 2

c. 41.375_{10} to base 4

d. 10011101.0101_2 to base 10

Exercise 2. Convert each of the following numbers to 8-bit signed magnitude, 8-bit one's complement, 8-bit two's complement and 8-bit excess 128 formats.

a. $(-122)_{10}$

b. $(-31)_{10}$

c. $(-16)_{10}$

d. 127_{10}

Exercise 3. Find the decimal equivalents for the following 8-bit two's complement numbers.

a. 1000 0001

b. 0111 1011

c. 1111 0001

d. 0010 1010

Exercise 4. Perform two's complement addition on the following pairs of numbers. In each case, indicate whether an overflow has occured.

a. 1110 1011 + 0110 1001

b. 1110 1011 + 1111 1111

c. 1000 1100 + 1100 0001

d. 0111 1001 + 0000 1001

Next Time

- Multiplication
- Floating Point numbers
- ASCII code