CIS 314
Unsigned and signed Integer, floating point

Prof. Michel A. Kinsy
Unsigned Integer

- It is what we covered in last lecture
- An unsigned integer containing \( n \) bits can have a value between 0 and \( 2^n - 1 \)
- Memory addresses are always represented by unsigned integers
- Given the binary number 11001:
  \[
  \begin{align*}
  \text{\textbullet} & \quad = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
  \text{\textbullet} & \quad = 16 + 8 + 0 + 0 + 1 \\
  \text{\textbullet} & \quad = 25
  \end{align*}
  \]
Signed Integer Representation

• Positive value representation alone are not the problem

• Negative value representation is the challenge
  ‣ To represent negative values, computer systems allocate the high-order bit to indicate the sign of a value
  ‣ The high-order bit is the leftmost bit in a binary number
  ‣ It is also called the most significant bit
  ‣ The remaining bits contain the value of the number
Signed Integer Representation

- There are three ways in which signed binary numbers may be expressed:
  - Signed magnitude
  - One’s complement
  - Two’s complement
- In an 8-bit binary number, signed magnitude representation places the absolute value of the number in the 7 bits to the right of the sign bit
  - $4 = 0 \ 0000100$
  - $-4 = 1 \ 0000100$
Signed magnitude Representation

- Arithmetic operations on signed magnitude numbers are performed inside the computer in much the same way as humans carry out pencil and paper arithmetic.

- Sum of 74 and 46
  - 1- Convert 74 and 46 to binary
  - 2- Arrange as a sum but separate the sign bits from the magnitude bits

\[
\begin{array}{c}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 + 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
= 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0
\end{array}
\]
Signed magnitude Representation

• In this example, two values were picked such as the sum would fit into seven bits

• If that is not the case, there is a problem

• Sum of 102 and 46
  ‣ The carry from the seventh bit overflows and it is discarded
  ‣ Erroneous result: 102 + 46 = 36

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 + 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0
\end{array}
\]
Signed magnitude Representation

- Signed magnitude representation is easy for people to understand
  - But it requires complicated computer hardware
  - Another disadvantage of signed magnitude is that it allows two different representations for zero: positive zero and negative zero

- So computers systems uses complement systems for numeric value representation
One’s complement

• With one’s complement addition, the carry bit is *carried around* and added to the sum

• Sum of 48 and –19
  ‣ 19 in binary is 00010011
  ‣ -19 in one’s complement is 11101100
  ‣ Answer is 29

\[
\begin{align*}
00110000 & \quad + \quad 11101100 \\
\quad = & \quad 00011100 \\
00011101 & \quad + \quad 1 \\
& \quad 00011101
\end{align*}
\]
One’s complement

• It amounts to little more than flipping the bits of a binary number

• In an 8-bit binary number using One’s complement
  - 4 = 0 0000100
  - -4 = 1 1111011

• In one’s complement, as with signed magnitude, negative values are indicated by a 1 in the high order bit
  - Complement systems are useful because they eliminate the need for subtraction
One’s complement

• Although the *end carry around* adds some complexity, one’s complement is simpler to implement in hardware than signed magnitude.

• But it still has the disadvantage of having two different representations for zero: positive zero and negative zero

• Two’s complement solves this problem
Two’s complement

• To express a value in two’s complement:
  • If the number is positive, convert it to binary and it is done
  • If the number is negative, find the one’s complement of the number and then add 1

- 4  = 0 0000100
- -4  = 1 1111100
Two’s complement

• With Two’s complement arithmetic

• Just add the two binary numbers

• Discard any carries emitting from the high order bit

• Sum of 48 and –19
  
  ‣ 19 in binary is 00010011
  ‣ -19 in one’s complement is 11101100
  ‣ -19 in two’s complement is 11101101
  ‣ Answer is 29
Overflow

- In a computing system, resources are finite
- There is always the risk that the result of a calculation becomes too large to be stored in the computer
- Overflow can not always be prevented
- It can always be detected
- In complement arithmetic, an overflow condition is easy to detect.
Overflow

• Using two’s complement binary arithmetic, the sum of 104 and 46 is

\[
\begin{array}{c}
0 1 1 0 1 0 0 0 \\
0 0 1 0 1 1 1 0 \\
\hline
1 0 0 1 0 1 1 0
\end{array}
\]

• The nonzero carry from the seventh bit overflows into the sign bit, resulting in an erroneous value: 
104 + 46 = -106
Overflow

• Good programmers stay alert for it
• Rule for detecting signed two’s complement overflow
  ‣ Carry in and carry out of the sign bit are different
• Rule for detecting unsigned number overflow
  ‣ There is carry out of the leftmost bit
  ‣ $1111 + 1 = 0000$
## Representation Range

### 3 bits
- **Signed:** -3, 3
- **1’s:** -3, 3
- **2’s:** -4, 3

### 6 bits
- **Signed:** -31, 31
- **1’s:** -31, 31
- **2’s:** -32, 31

### 8 bits
- **Signed:** -127, 127
- **1’s:** -127, 127
- **2’s:** -128, 127

### 5 bits
- **Signed:** -15, 15
- **1’s:** -15, 15
- **2’s:** -16, 15

### 8 bits
- **Signed:** -127, 127
- **1’s:** -127, 127
- **2’s:** -128, 127

---

### Formula for calculating the range

Signed & 1’s: \(-(2^{n-1} - 1), (2^{n-1} - 1)\)

2’s: \(-2^{n-1}, (2^{n-1} - 1)\)
Floating-Point Representation

• Signed magnitude, one’s complement, and two’s complement representations alone are not useful in scientific or business applications.

• Real number values over a wide range.

• Numerals to the right of a base point represent negative powers of the base.

\[
0.11_2 = 1 \times 2^{-1} + 1 \times 2^{-2} = 0.5 + 0.25 = 0.75
\]
Floating-Point Representation

• Converting 0.8125 to binary

• This method also works with any base. Just use the target base as the multiplier.
  ‣ 0.8125 * 2 = 1.6250
  ‣ 0.6250 * 2 = 1.2500
  ‣ 0.2500 * 2 = 0.5000
  ‣ 0.5000 * 2 = 1.0000

Reading from top to bottom is: 0.8125_{10} = 0.1101_{2}

• Conversion is done when the product is zero or the desired number of binary places is reached
Floating-Point Representation

• Computer systems use a form of scientific notation for floating-point representation

• Numbers written in scientific notation have three components:
  
  • Sign  Mantissa  Exponent

  + 1.5 x 10^{-1}

• Computer representation of a floating-point number consists of three fixed-size fields:

  Sign  Exponent  Significand
Floating-Point Representation

- The one-bit sign field is the sign of the stored value.
- The size of the exponent field determines the range of values that can be represented.
- The size of the significand determines the precision of the representation.

| Sign | Exponent | Significand |
Floating-Point Representation

- The IEEE-754 single precision floating point standard uses an 8-bit exponent and a 23-bit significand
- The IEEE-754 double precision standard uses an 11-bit exponent and a 52-bit significand
Floating-Point Representation

- The significand of a floating-point number is always preceded by an implied binary point
- The significand always contains a fractional binary value
- The exponent indicates the power of 2 to which the significand is raised
Floating-Point Representation

• Express $32_{10}$ in the simplified 14-bit floating-point representation

• $32 = 2^5 = 1.0 \times 2^5 = +0.1 \times 2^6$

• Exponent field = $110_2 = 6_{10}$

• Significand field 1

• How do we express fractional numbers $0.5 = 2^{-1}$?
Floating-Point Representation

• To provide for negative exponents, a biased exponent is used

• The IEEE-754 single precision floating point standard uses bias of 127 over its 8-bit exponent
  ‣ An exponent of 255 indicates
  ‣ Infinity if significand is zero
  ‣ NaN, “not a number,” often used to flag an error condition if the significand is nonzero

• The double precision standard has a bias of 1023 over its 11-bit exponent
Floating-Point Representation

• Sum of $12_{10}$ and $1.25_{10}$ using the 14-bit floating-point representation
  ‣ $12_{10} = 0.1100 \times 2^4$
  ‣ $1.25_{10} = 0.101 \times 2^1 = 0.000101 \times 2^4$
  ‣ Thus, the sum is $0.110101 \times 2^4$
Floating-Point Representation

• Using the same number of bits, unsigned integers can express twice as many values as signed numbers.

• Using two’s complement allows for one type of hardware/process to add both signed and unsigned numbers.

• IEEE-754 floating point standards allow two representations for zero:
  ‣ Programmers should avoid testing a floating-point value for equality to zero.
  ‣ Negative zero does not equal positive zero.
Next class

• Introduction to programming with C