# COMP311: *COMPUTER ORGANIZATION!*

Lecture 2: 211 Review

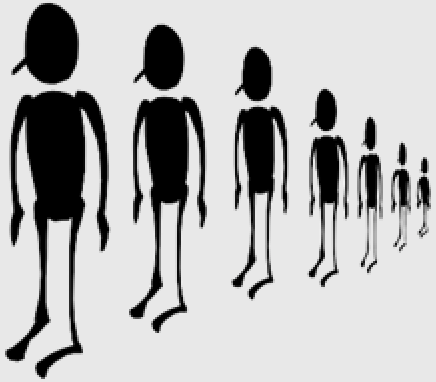
tinyurl.com/comp311-fa25

# Encoding Positive Integers

Encode positive integers as a ***sequence of bits.***

Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an n-bit number encoded in this fashion is given by the following formula:

$$v = \sum_{i=0}^{n-1} 2^i b_i$$



| $2^{15}$ | $2^{14}$ | $2^{13}$ | $2^{12}$ | $2^{11}$ | $2^{10}$ | $2^9$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

# Some Important Bits

- You are going to have to get accustomed to working in binary, but it will be helpful throughout your career as a computer scientist.
- Some good ones to know
  - *The first 10 powers of 2*
  - *The prefixes for powers of 2 that are powers of 10*

# Review: Binary Addition and Overflow

Q3 from last time

```
   1010₂
+  0101₂
---------
   1111      no
```

```
 1111
   1001₂
+  0111₂
---------
   0000      yes
```

```
 111
   1010₂
+  0111₂
---------
   0001      yes
```

```
  11
   0110₂
+  0110₂
---------
   1100      no
```

# Recall: Signed Integers

- One strategy is to ***encode the sign*** of the integer ***using one bit***.
  - *Conventionally, the most significant bit is used for the sign.*
- This encoding of signed integers is called “SIGNED MAGNITUDE”

$$v = -1^S \sum_{i=0}^{n-2} 2^i \, b_i$$

| S | $2^{14}$ | $2^{13}$ | $2^{12}$ | $2^{11}$ | $2^{10}$ | $2^{9}$ | $2^{8}$ | $2^{7}$ | $2^{6}$ | $2^{5}$ | $2^{4}$ | $2^{3}$ | $2^{2}$ | $2^{1}$ | $2^{0}$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |

- 2024


- The Good
  - *Easy toe negate, easy to take absolute value*
- The Bad
  - *Two ways to represent “0”, +0 and -0*
  - *Add/subtract is complicated; depends on the signs*

# Recall: 2's Complement Notation

- The 2's complement representation for signed integers is the most commonly used signed-integer representation.
- It is a simple modification of unsigned integers where the most significant bit is a negative power of 2.

$$v = -2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i$$

| $2^{15}$ | $2^{14}$ | $2^{13}$ | $2^{12}$ | $2^{11}$ | $2^{10}$ | $2^9$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |

**Still a "sign bit"**
**(It must be "1" for the number to < 0)**

-32768
+2024
-30744

# Why 2’s complement?

- In the two’s complement representation for signed integers, the same binary “addition procedure” (mod $2^n$) works for adding any combination of positive and negative numbers.
- Don’t need a separate “subtraction circuitry” (carries only, no borrows)
  - *The “addition procedure” also handles unsigned numbers!*
  - *In 2’s complement adding is "adding" regardless of operand signs.*
  - *You NEVER need to subtract when you use 2’s-complement.*

# 2's complement tricks

- **Negation:** changing the sign of a number
  - *Invert every bit (i.e. 1 → 0, 0 → 1) and add 1*

Example: $42_{10} = 000000101010_2$
$-42_{10} = 111111010101_2 + 1 =$
$111111010110_2$

- **Sign-Extension** - aligning different sized 2's complement integers (for example when adding an 8-bit number to a 32-bit number)
  - *Simply copy the sign bit into higher positions*

Example: 16-bit version of 42: $42_{10} = 0000000000101010_2$
16-bit version of -42: $-42_{10} = 1111111111010110_2$

# Two’s Complement

Take 5 minutes to answer question 4 on your worksheets!

Q4 from last time

# Two's Complement

4.1: Represent -15 in two's complement with 7 bits

Unsigned: `0b 000 1111`

Flip bits: `0b 111 0000`

Add one: `0b 111 0001`

4.2: What is the minimum number of bits needed to represent 20 in two's complement

`0b 010100`

6 bits

4.3: Convert the following two's complement number to decimal: 0b 1111 1110

Flip bits: `0b 0000 0001`

Add one: `0b 0000 0010` → `-2`

4.4: Convert the following two's complement number to decimal: 0b 0011 1000

Positive, so no need to flip bits

56

# Two's Complement Overflow

Q5 from last time

Take 5 minutes to answer question 5 on your worksheets!

# Two's Complement Overflow

5.1

$$\begin{array}{r} 111 \\ 0111_2 \\ +\ 0101_2 \\ \hline 1100 \end{array} \rightarrow -4 \qquad \begin{array}{r} 7 \\ +\ 5 \\ \hline 12 \end{array}$$

Overflow

5.3

$$\begin{array}{r} 111 \\ 1111_2 \\ +\ 0110_2 \\ \hline 0101 \end{array} \rightarrow 5 \qquad \begin{array}{r} -1 \\ +\ 6 \\ \hline 5 \end{array}$$

No Overflow

5.2

$$\begin{array}{r} 1 \\ 1000_2 \\ +\ 1001_2 \\ \hline 0001 \end{array} \rightarrow 1 \qquad \begin{array}{r} -8 \\ +\ -7 \\ \hline -15 \end{array}$$

Overflow

$$\begin{array}{r} 1111 \\ 1101_2 \\ +\ 1011_2 \\ \hline 1000 \end{array} \rightarrow -8 \qquad \begin{array}{r} -3 \\ +\ -5 \\ \hline -8 \end{array}$$

No Overflow


12

# Two’s Complement Overflow

- Overflow: when the result of an operation cannot be represented in the given number of bits

| Adding | Overflow occurs when |
|---|---|
| two positive numbers | the result is negative |
| two negative numbers | the result is positive |
| two numbers of opposite signs | will never occur |

# Two’s Complement Subtraction

$$\begin{array}{r} -10 \\ -\quad 4 \\ \hline -14 \end{array} \qquad \begin{array}{r} -10 \\ +\quad -4 \\ \hline -14 \end{array}$$

# Two's Complement Subtraction

$$\begin{array}{r} -10 \\ -\quad 4 \\ \hline -14 \end{array} \qquad \begin{array}{r} -10 \\ +\quad -4 \\ \hline -14 \end{array} \qquad \begin{array}{r} 10110_2 \\ -\quad 00100_2 \\ \hline \end{array}$$

# Two's Complement Subtraction

$$\begin{array}{r} -10 \\ -\quad 4 \\ \hline -14 \end{array} \qquad \begin{array}{r} -10 \\ +\quad -4 \\ \hline -14 \end{array}$$

$$\begin{array}{r} 10110_2 \\ -\ 00100_2 \\ \hline \end{array} \qquad \begin{array}{r} 10110_2 \\ +\ 11100_2 \\ \hline 10010_2 \end{array}$$

# Two's Complement Subtraction

$$\begin{array}{r} -10 \\ -\quad 4 \\ \hline -14 \end{array} \qquad \begin{array}{r} -10 \\ +\quad -4 \\ \hline -14 \end{array} \qquad \begin{array}{r} 10110_2 \\ -\ 00100_2 \\ \hline \end{array} \qquad \begin{array}{r} 10110_2 \\ +\ 11100_2 \\ \hline 10010_2 \end{array}$$

---

$$\begin{array}{r} -10 \\ -\quad -4 \\ \hline -6 \end{array} \qquad \begin{array}{r} -10 \\ +\quad 4 \\ \hline -6 \end{array} \qquad \begin{array}{r} 10110_2 \\ -\ 11100_2 \\ \hline \end{array} \qquad \begin{array}{r} 10110_2 \\ +\ 00100_2 \\ \hline 11010_2 \end{array}$$

# Two's Complement Subtraction

Q1

$$\begin{array}{r} 00101_2 \\ -\ 00111_2 \\ \hline \end{array}$$

Q2

$$\begin{array}{r} 10101_2 \\ -\ 10110_2 \\ \hline \end{array}$$

Q3

$$\begin{array}{r} 11111_2 \\ -\ 01000_2 \\ \hline \end{array}$$

Q4

$$\begin{array}{r} 01110_2 \\ -\ 11100_2 \\ \hline \end{array}$$

# Bitwise Operations

- Apply the operation to each individual bit position

$$\sim 1010_2 = 0101_2$$

$$\begin{array}{r} 1010_2 \\ \& \; 1100_2 \\ \hline 1000_2 \end{array}$$

$$\begin{array}{r} 1010_2 \\ | \; 1100_2 \\ \hline 1110_2 \end{array}$$

$$\begin{array}{r} 1010_2 \\ \wedge \; 1100_2 \\ \hline 0110_2 \end{array}$$

# Bitwise Operations: Fill in the blank!

Q1

- Fill in the following blanks using the options below. n is an integer. There are multiple correct answers for each question.
  - *A. n ___ ________ = n*
  - *B. n ___ ________ = 0b 00…00*
  - *C. n ___ ________ = 0b 11…11*

| **First blank** | **Second blank** |
| --- | --- |
| AND | n |
| OR | ~n |
| XOR | 0b 00…00 |
| | 0b 11…111 |

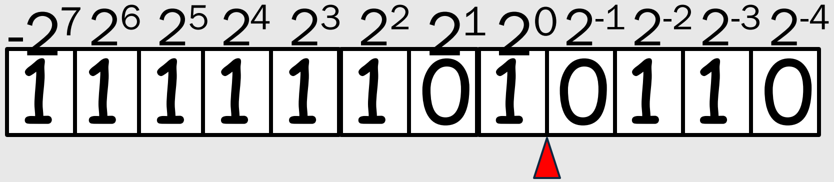
# Bitwise Operations: Fill in the blank!

- *A. n ____ __________ = n*
  - OR 0b 00…00
  - AND 0b 11…1
  - XOR 0b 00…00
  - AND n
  - OR n
- *B. n ____ __________ = 0b 00…00*
  - AND 0b 00…00
  - AND ~n
  - XOR n
- *C. n ____ __________ = 0b 11…11*
  - OR 0b11…11
  - OR ~n
  - XOR ~n

| First blank | Second blank |
|---|---|
| AND | n |
| OR | ~n |
| XOR | 0b 00…00 |
| | 0b 11…111 |

# Fixed-Point Numbers

- You can always assume that the boundary between 2 bits is a “binary point”.
- If you **align** binary points between addends, there is no effect on how operations are performed.




$$
\begin{aligned}
11111101.0110 &= -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} \\
&= -128 + 64 + 32 + 16 + 8 + 4 + 1 + 0.25 + 0.125 \\
&= -2.625
\end{aligned}
$$

**OR**

$$
\begin{aligned}
11111101.0110 &= -42 \times 2^{-4} \\
&= -42 \;/\; 16 \\
&= -2.625
\end{aligned}
$$

# Repeated Binary Fractions

- Not all fractions can be represented exactly using a **finite representation.**
  - *Ex: 1/3 → 0.333333....*
- In binary, many fractions that you've grown attached to require an infinite number of bits to represent exactly.

Example: $1/10 = 0.1_{10} = 0.00011..._{2} = 0.19..._{16}$

$1/5 = 0.2_{10} = 0.0011..._{2} = 0.3..._{16}$

$1/3 = 0.3_{10} = 0.01..._{2} = 0.5..._{16}$

# Finite Representations!

- **Everything** that a realizable computer does is limited by a finite set of bits.
- You may have grown used to infinite digits in math courses…

```
        ...00000000042.0000000000...
        ...00000000000.0000000000...001000
10000000...00000000000.0
```

- …However, the concept an infinite supply of zero digits is conceptually elegant, but difficult to physically implement

# Overflow: Side Effect of being Finite

- Overflow: when the result of an operation cannot be represented in the given number of bits

1. $32767_{10} + 1_{10} = -32768_{10}$

$$
\begin{array}{r}
0111\ 1111\ 1111\ 1111_2 \\
0000\ 0000\ 0000\ 0001_2 \\
\hline
1000\ 0000\ 0000\ 0000_2
\end{array}
$$

2. $-20000_{10} - 20000_{10} = 25536_{10}$

$$
\begin{array}{r}
1011\ 0001\ 1110\ 0000_2 \\
+\quad 1011\ 0001\ 1110\ 0000_2 \\
\hline
1\ 0110\ 0011\ 1100\ 0000_2
\end{array}
$$

3. $-32768_{10} = -32768_{10}$

A certain number can't be negated

$$
\begin{array}{r}
1000\ 0000\ 0000\ 0000_2 \\
0111\ 1111\ 1111\ 1111_2 \\
+\quad 0000\ 0000\ 0000\ 0001_2 \\
\hline
1000\ 0000\ 0000\ 0000_2
\end{array}
$$

Flip bits first, then add one 1 negate…

# 211 REVIEW: MEMORY

# Storing Data in Memory

1 byte = 8 bits

Each of these rows can store one byte of data

# Storing Data in Memory

1 byte = 8 bits

Each of these rows can store one byte of data

| |
|---|
| 0b0000 0101 |
| 0b0000 0100 |
| 0b1111 1110 |
| |
| |
| |
| |
| |
| |
| |
| |
| |

Let's say we want to store the number 5 in the top row, 4 in the next row, and -2 in the next row.

# Byte Addresses

1 byte = 8 bits

| Byte Address | |
|---|---|
| 0 | 0b0000 0101 |
| 1 | 0b0000 0100 |
| 2 | 0b1111 1110 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |

To reference each of these locations, we use something called a byte address.

Note that these addresses are not stored anywhere!