# COMPSCI 210 S2 2024
# Computer Organisation

2 Data Representation

# Overview

- Binary digital systems
- Basic data types
- Operations
  - Logic
  - Arithmetic

# Data Representation in Computer

- At the lowest level, a computer is an electronic machine
  - Works by controlling the flow of electrons
  - Easy to recognize two conditions:
    - 1. presence of a voltage – we'll call this state "1"
    - 2. absence of a voltage – we'll call this state "0"
- How Computers work?
  - https://www.youtube.com/watch?v=6b4uvXBCI7w

# Binary Digital System

- Basic unit of information is the binary digit, or bit
- Values with more than two states require multiple bits
- A collection of two bits has four possible states:
  - 00, 01, 10, 11
- A collection of three bits has eight possible states:
  - 000, 001, 010, 011, 100, 101, 110, 111

With **n** bits, can represent **$2^n$** different values.

# Data Types

- In a computer system, we need a **representation** of data and **operations** that can be performed on the data by the machine instructions or the computer language.
- This combination of *representation* + *operations* is known as a **data type**.

| Type | Representation | Operations |
|---|---|---|
| Unsigned integers | binary | add, multiply, etc. |
| Signed integers | 2's complement binary | add, multiply, etc. |
| Real numbers | IEEE floating-point | add, multiply, etc. |
| Text characters | ASCII | input, output, compare |

# Unsigned Integers

## Non-positional notation (unary)

- Could represent a number (“5”) with a string of ones (“11111”).
- Problems?

## Weighted positional notation (binary)

- Like decimal numbers: “329.”
- “3” is worth 300, because of its position, while “9” is only worth 9.
- Since only 0 and 1 digits, use a binary (base-two) number system.

329

$10^2$ $10^1$ $10^0$

$\boxed{3\times100 + 2\times10 + 9\times1 = 329}$

*most significant* 101 *least significant*

$2^2$ $2^1$ $2^0$

$\boxed{1\times4 + 0\times2 + 1\times1 = 5}$

# Unsigned Integers

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

| $2^2$ | $2^1$ | $2^0$ | value |
|---|---|---|---|
| 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 |

# Exercise 2.1 Q1.1.

What is the range of a 6-bit unsigned integer?

- 0 - 64
- 0 - 63
- 0 - 31
- 0 - 32
- None of the others

# Exercise 2.1 Q1.1.

What is the range of a 6-bit unsigned integer?

- 0 - 64
- 0 - 63 ✔
- 0 - 31
- 0 - 32
- None of the others

# Exercise 2.1 Q1.2.

What is the range of an 8-bit unsigned integer?

- 0 - 512
- 0 - 511
- 0 - 256
- 0 - 255
- None of the others

# Exercise 2.1 Q1.2.

What is the range of an 8-bit unsigned integer?

- 0 - 512
- 0 - 511
- 0 - 256
- 0 - 255 ✔
- None of the others

# Signed Integers

- How to represent positive (+) and negative (–)?
- With n bits, we have $2^n$ distinct values
  - Assign about half to positive integers (1 through $(2^{n-1})-1$) and about half to negative ($-(2^{n-1})+1$ through -1)
  - That leaves two values: one for 0, and one extra
- Three different encoding schemes
  - Signed-magnitude.
  - 1's complement.
  - 2's complement.

# Signed-Magnitude

- Leftmost bit represents the **sign** of the number.
  - 0 = positive
  - 1 = negative
- Remaining bits represent the **magnitude** of the number using the binary notation used for unsigned integers.
- Examples of 5-bit signed-magnitude integers:
  - 00101 = +5 (sign = 0, magnitude = 0101)
  - 10101 = −5 (sign = 1, magnigude = 0101)
  - 01101 = +13 (sign = 0, magnitude = 1101)
  - 10010 = −2 (sign = 1, magnitude = 0010)

# 1's Complement

- To get a negative number, start with a positive number (with zero as the leftmost bit) and flip all the bits -- from 0 to 1, from 1 to 0.
- Leftmost bit indicates sign: 0=positive, 1=negative
- Examples: 5-bit 1's complement integers:

```
00101  (5)        01001  (9)
11010  (-5)       10110  (-9)
```

# 2’s Complement

- Problems with sign-magnitude and 1’s complement
  - Two representations of zero (+0 and –0)
  - Arithmetic logic circuits are complex
    - How to add two sign-magnitude numbers? e.g., try 2 + (-3)
    - How to add two one’s complement numbers? e.g., try 4 + (-3)
- Two’s complement representation developed to make logic 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's Complement – Method 1

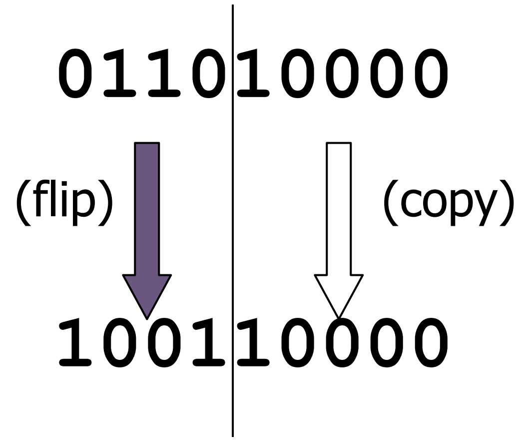
- If number is positive or zero
  - Normal (unsigned) binary representation, zeroes in upper bit(s)
- If number is negative
  - Start with positive number
  - Flip every bit (i.e., take the one's complement)
  - Then add one

```
  00101 (5)            10111 (-9)
  11010 (1's comp)     01000 (1's comp)
+     1              +     1
-------              -------
  11011 (-5)           01001 (9)
```

# 2’s Complement – Method 2

- 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
```



# 2’s Complement

- With n bits, represent values from $-2^{n-1}$ to $+2^{n-1}-1$

| 4-bit 2's complement | value | 4-bit 2's complement | value |
|---|---|---|---|
| 0000 | 0 | | |
| 0001 | 1 | 1111 | –1 |
| 0010 | 2 | 1110 | –2 |
| 0011 | 3 | 1101 | –3 |
| 0100 | 4 | 1100 | –4 |
| 0101 | 5 | 1011 | –5 |
| 0110 | 6 | 1010 | –6 |
| 0111 | 7 | 1001 | –7 |
| | | 1000 | –8 |

- NOTE: All positive number start with 0, all negative numbers start with 1.

## 2.2 Data Type

# Types of representations of integers

| Representation | Value Represented | | | |
|---|---|---|---|---|
| | Unsigned | Signed Magnitude | 1's Complement | 2's Complement |
| 00000 | 0 | 0 | 0 | 0 |
| 00001 | 1 | 1 | 1 | 1 |
| 00010 | 2 | 2 | 2 | 2 |
| 00011 | 3 | 3 | 3 | 3 |
| 00100 | 4 | 4 | 4 | 4 |
| 00101 | 5 | 5 | 5 | 5 |
| 00110 | 6 | 6 | 6 | 6 |
| 00111 | 7 | 7 | 7 | 7 |
| 01000 | 8 | 8 | 8 | 8 |
| 01001 | 9 | 9 | 9 | 9 |
| 01010 | 10 | 10 | 10 | 10 |
| 01011 | 11 | 11 | 11 | 11 |
| 01100 | 12 | 12 | 12 | 12 |
| 01101 | 13 | 13 | 13 | 13 |
| 01110 | 14 | 14 | 14 | 14 |
| 01111 | 15 | 15 | 15 | 15 |
| 10000 | 16 | –0 | –15 | –16 |
| 10001 | 17 | –1 | –14 | –15 |
| 10010 | 18 | –2 | –13 | –14 |
| 10011 | 19 | –3 | –12 | –13 |
| 10100 | 20 | –4 | –11 | –12 |
| 10101 | 21 | –5 | –10 | –11 |
| 10110 | 22 | –6 | –9 | –10 |
| 10111 | 23 | –7 | –8 | –9 |
| 11000 | 24 | –8 | –7 | –8 |
| 11001 | 25 | –9 | –6 | –7 |
| 11010 | 26 | –10 | –5 | –6 |
| 11011 | 27 | –11 | –4 | –5 |
| 11100 | 28 | –12 | –3 | –4 |
| 11101 | 29 | –13 | –2 | –3 |
| 11110 | 30 | –14 | –1 | –2 |
| 11111 | 31 | –15 | –0 | –1 |

# Exercise 2.2 Q2.1.

What is the 5-bit two's complement representation of -9?

- 11001
- 10110
- 01001
- 10111

# Exercise 2.2 Q2.1.

What is the 5-bit two's complement representation of -9?

- 11001
- 10110
- 01001
- 10111 ✔

# Exercise 2.2 Q2.2.

What is the 8-bit signed-magnitude representation of -26?

What is the 8-bit one's complement representation of -26?

What is the 8-bit two's complement representation of -26?

# Exercise 2.2 Q2.2.

What is the 8-bit signed-magnitude representation of -26?

- 1001 1010

What is the 8-bit one's complement representation of -26?

- 1110 0101

What is the 8-bit two's complement representation of -26?

- 1110 0110

# Hexadecimal Notation

- It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead.
  - Fewer digits -- four bits per hex digit
  - Less error prone – easy to corrupt long string of 1's and 0's

| Binary (base 2) | Hex (base 16) | Decimal (base 10) |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 4 | 4 |
| 0101 | 5 | 5 |
| 0110 | 6 | 6 |
| 0111 | 7 | 7 |

| Binary (base 2) | Hex (base 16) | Decimal (base 10) |
|---|---|---|
| 1000 | 8 | 8 |
| 1001 | 9 | 9 |
| 1010 | A | 10 |
| 1011 | B | 11 |
| 1100 | C | 12 |
| 1101 | D | 13 |
| 1110 | E | 14 |
| 1111 | F | 15 |

# Conversion of Hexadecimal Notation

- Converting from Hexadecimal to Binary
  - Every hex digit can be converted to four binary bits

| 3 | A | 8 | F | 4 | D | 7 |
|---|---|---|---|---|---|---|
| 0011 | 1010 | 1000 | 1111 | 0100 | 1101 | 0111 |

# Conversion of Hexadecimal Notation

- Converting from Binary to Hexadecimal
  - Every four bits is a hex digit
    - Start grouping from right-hand side
    - Sign-extend if needed

| 0011 | 1010 | 1000 | 1111 | 0100 | 1101 | 0111 |
|---|---|---|---|---|---|---|
| ⇩ | ⇩ | ⇩ | ⇩ | ⇩ | ⇩ | ⇩ |
| **3** | **A** | **8** | **F** | **4** | **D** | **7** |

# Conversion of Hexadecimal Notation

- Converting from Hexadecimal to Decimal
  - Add powers of 16 that have non-zero digit in the corresponding digit positions

$$
\begin{aligned}
X &= A043_{16} \\
&= 10*16^3+4*16^1+3*16^0 = 40960+64+3 \\
&= 41027_{10}
\end{aligned}
$$

# Exercise 2.3 Q3.1.

Convert $A2_{16}$ to a decimal value.

- 162
- 12
- 20
- 102

# Exercise 2.3 Q3.1.

Convert $A2_{16}$ to a decimal value.

- 162 ✔
- 12
- 20
- 102

# Exercise 2.3 Q3.2.

Convert $AC_{16}$ to a binary value.

- 1010 1100
- 172
- 1001 1100
- 1001 1001

# Exercise 2.3 Q3.2.

Convert $AC_{16}$ to a binary value.

- 1010 1100 ✔
- 172
- 1001 1100
- 1001 1001

# Representation of non-Integers

- Text, strings, image, sound
- Fractions
- Scientific notation/Floating point representation

# ASCII Representation

- American Standard Code for Information Interchange (ASCII)
  - Developed from telegraph codes, alternative to IBM's Extended Binary Coded Decimal Interchange Code (EBCDIC) in 1960s
  - Printable and non-printable (ESC, DEL, …) characters (0-127)
  - Limited set of characters – many characters missing, especially language-specific
  - Extended ASCII characters
  - Many national "standards" developed

# ASCII Representation

- ASCII Characters
  - Maps characters to an 8-bit code (with 7 bits used for encoding)
    - Both printable and non-printable (ESC, DEL, …) characters

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | nul | 10 | dle | 20 | sp | 30 | 0 | 40 | @ | 50 | P | 60 | ` | 70 | p |
| 01 | soh | 11 | dc1 | 21 | ! | 31 | 1 | 41 | A | 51 | Q | 61 | a | 71 | q |
| 02 | stx | 12 | dc2 | 22 | " | 32 | 2 | 42 | B | 52 | R | 62 | b | 72 | r |
| 03 | etx | 13 | dc3 | 23 | # | 33 | 3 | 43 | C | 53 | S | 63 | c | 73 | s |
| 04 | eot | 14 | dc4 | 24 | $ | 34 | 4 | 44 | D | 54 | T | 64 | d | 74 | t |
| 05 | enq | 15 | nak | 25 | % | 35 | 5 | 45 | E | 55 | U | 65 | e | 75 | u |
| 06 | ack | 16 | syn | 26 | & | 36 | 6 | 46 | F | 56 | V | 66 | f | 76 | v |
| 07 | bel | 17 | etb | 27 | ' | 37 | 7 | 47 | G | 57 | W | 67 | g | 77 | w |
| 08 | bs | 18 | can | 28 | ( | 38 | 8 | 48 | H | 58 | X | 68 | h | 78 | x |
| 09 | ht | 19 | em | 29 | ) | 39 | 9 | 49 | I | 59 | Y | 69 | i | 79 | y |
| 0a | nl | 1a | sub | 2a | * | 3a | : | 4a | J | 5a | Z | 6a | j | 7a | z |
| 0b | vt | 1b | esc | 2b | + | 3b | ; | 4b | K | 5b | [ | 6b | k | 7b | { |
| 0c | np | 1c | fs | 2c | , | 3c | < | 4c | L | 5c | \ | 6c | l | 7c | \| |
| 0d | cr | 1d | gs | 2d | - | 3d | = | 4d | M | 5d | ] | 6d | m | 7d | } |
| 0e | so | 1e | rs | 2e | . | 3e | > | 4e | N | 5e | ^ | 6e | n | 7e | ~ |
| 0f | si | 1f | us | 2f | / | 3f | ? | 4f | O | 5f | _ | 6f | o | 7f | del |

# Exercise 2.4

- Q4.1. What is the ASCII character with the value 0x37 ($37_{16}$) while 0x30 is ‘0’?

- Q4.2. What is the ASCII character with the value ‘a’ + 5?

# Logic Operations

- The three basic logical (or Boolean) operations are:
  - AND
  - OR
  - NOT

# Logic Operations

- Bit-wise logical operations
  - Operations on logical TRUE or FALSE
    - Two states -- takes one bit to represent: TRUE=1, FALSE=0
  - View n-bit number as a collection of n logical values
    - Operation applied to each bit independently

| A | B | A AND B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| A | B | A OR B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| A | NOT A |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Logic Operations

- Bit-wise logical operations
  - AND
    - Useful for clearing bits
      - AND with zero = 0
      - AND with one = no change
  - OR
    - Useful for setting bits
      - OR with zero = no change
      - OR with one = 1
  - NOT
    - Unary operation -- one argument
      - Flips every bit

```
     11000101
AND  00001111
     --------
     00000101

     11000101
OR   00001111
     --------
     11001111

NOT  11000101
     --------
     00111010
```

# Operations

- Recall : a data type includes representation and operations
- We now have a good representation for signed integers (especially 2's complement) so, let's look at some arithmetic operations:
  - Addition
  - Subtraction
  - Sign Extension
- We'll also look at overflow conditions for addition

# Addition

- 2's complement addition is just binary addition.
  - Assume all integers have the same number of bits
  - Sign bit is treated the same as other bits
  - For now, assume that sum fits in n-bit 2's comp. representation
  - Ignore carry out

*Assuming 8-bit 2's complement numbers.*

```
  00001000 (8)            11110110 (-10)
+ 11111100 (-4)       +   11110111 (-9)
  --------                --------
  00000100 (4)            11101101 (-19)
```

# Subtraction

- Assume all integers have the same number of bits
- X – Y = X + (-Y)
  - "–Y" is obtained by calculating Y's 2's complement
- For now, assume that the result fits in n-bit 2's comp. representation
- Ignore carry out

*Assuming 8-bit 2's complement numbers.*

```
  00001000 (8)          11110110 (-10)
- 00000100 (4)        - 11110111 (-9)
  --------              --------
  00001000 (8)          11110110 (-10)
+ 11111100 (-4)       + 00001001 (9)
  --------              --------
  00000100 (4)          11111111 (-1)
```

# Sign Extension

- To add/subtract two numbers, we must represent them with the same number of bits
- If we just pad with zeroes on the left:

| 4-bit | | 8-bit | |
|---|---|---|---|
| 0100 | (4) | 00000100 | (still 4) |
| 1100 | (-4) | 00001100 | (12, not -4) |

- Instead, replicate the most significant (MS) bit -- the sign bit:

| 4-bit | | 8-bit | |
|---|---|---|---|
| 0100 | (4) | 00000100 | (still 4) |
| 1100 | (-4) | 11111100 | (still -4) |

# Overflow

- If operands are too big, then sum cannot be represented as an n-bit 2's comp. number
- For 2's complement, this can only happen if both numbers are positive or both numbers are negative.

```
   01000  (8)          11000  (-8)
+  01001  (9)        + 10111  (-9)
   -----               -----
   10001  (-15)        01111  (+15)
```

- We have overflow if:
  - Signs of both operands are the same, and
  - Sign of sum is different

# Summary

- On completion of this class, you are able to
  - Familiar with the binary representation
  - Familiar with the hexadecimal representation
  - Familiar with the ASCII representation
  - Carry out simple logical operations for binary values
  - Convert the number representation in computer
  - Calculate simple binary arithmetical and logical operations
- Reading:
  - Chapter 2 of textbook