

# CS 223

## Week 8 Lecture 1



# Agenda

Announcements

Course grade breakdown

Remaining binary topics

Bitwise operators

# Announcements

Quiz 02: Wed Oct 28 in lab

HW03: Due Sun Oct 25 before midnight

Today's topics:

- Remaining binary topics
- Bitwise operators

# Grade Breakdown

Final Exam: 30%

Quizzes: 35%

Homework: 15%

Labs: 15%

Attendance / Participation: 5%

# Visualizing Overflow

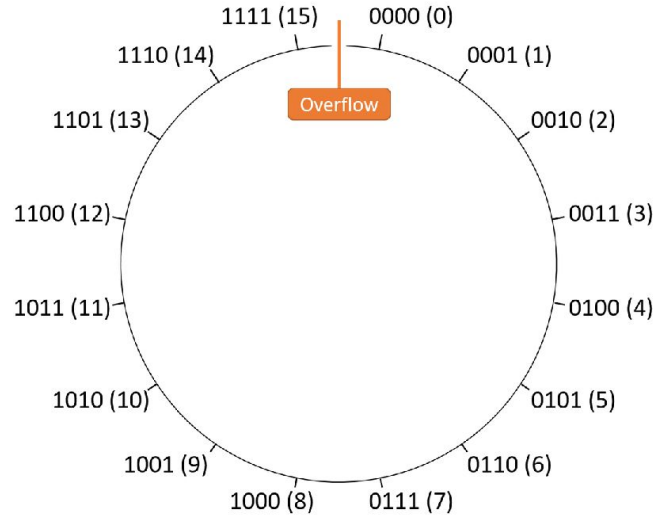


*Figure 3. An arrangement of four-bit unsigned values into a modular space. All arithmetic is modular with respect to  $2^4$  (16).*

# Binary addition

Possible outcomes when considering two binary digits and 1 carry digit

| Inputs  |         |          |  | Outputs |           |
|---------|---------|----------|--|---------|-----------|
| Digit A | Digit B | Carry in |  | Sum     | Carry out |
| 0       | 0       | 0        |  |         |           |
| 0       | 0       | 1        |  |         |           |
| 0       | 1       | 0        |  |         |           |
| 0       | 1       | 1        |  |         |           |
| 1       | 0       | 0        |  |         |           |
| 1       | 0       | 1        |  |         |           |
| 1       | 1       | 0        |  |         |           |
| 1       | 1       | 1        |  |         |           |

# Example: Add 8 + 3

Add 8 + 3 as unsigned 4-bit numbers

# Example: Add 3 + 3

Add 3 + 3 as unsigned 4-bit numbers

# Binary subtraction

Idea:  $X - Y$  is the same as  $X + (-Y)$

Therefore, negate the second operand and then add

Example: Compute  $7-3$  as 4 bit signed integers

# Signed extension

What happens when you perform an arithmetic operation on numbers with different sizes? **signed extension**

For *unsigned* values, prepend 0

For *signed* values, prepend the leftmost bit

# Example: Signed extension

Add 5 and -5 as 8 bit numbers

# Signed extension: example

|            |  |
|------------|--|
| 0b 1011    |  |
| 0b 1 1011  |  |
| 0b 11 1011 |  |

# Floating point

## Goals of representation

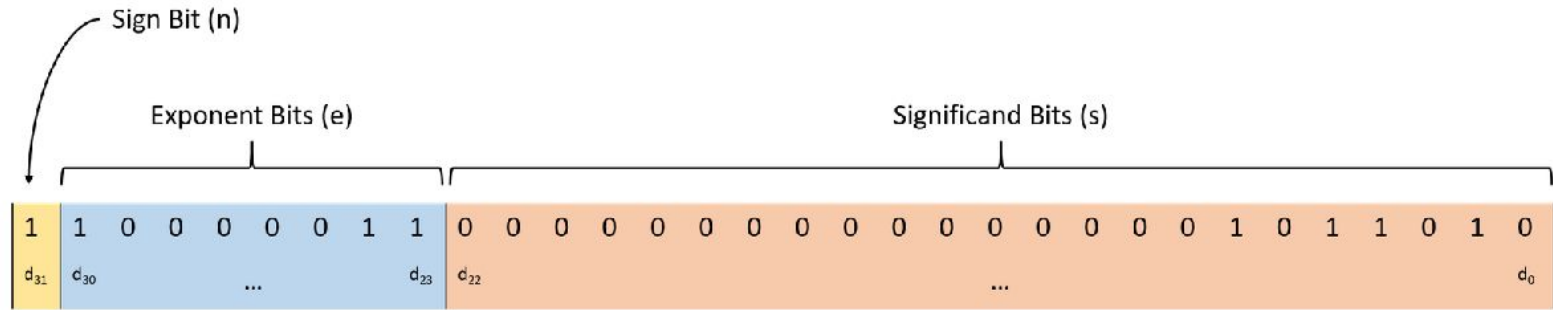
- Need to encode the sign

- Need to encode both very large and very small numbers

Based on scientific notation, where numbers are represented by a fraction and an exponent

Ex:  $12762 = 1.2762 \times 10^4$

# Floating point



Interpretation:  $(-1)^n * 2^{(e-127)} * 1.s$

*Figure 45. The 32-bit IEEE 754 floating-point standard*

NOTE:  $e-127$  called the **bias**, range is  $[-126,127]$

# Floating point example

What is the value of this binary floating point number in decimal?

1100 0001 1011 0100 0000 0000 0000 0000

# Bitwise operations: AND, OR, XOR, NOT

Table 1. The Results of Bitwise ANDing Two Values (A AND B)

| A | B | A & B |
|---|---|-------|
| 0 | 0 | 0     |
| 0 | 1 | 0     |
| 1 | 0 | 0     |
| 1 | 1 | 1     |

Table 3. The Results of Bitwise XORing Two Values (A XOR B)

| A | B | A ^ B |
|---|---|-------|
| 0 | 0 | 0     |
| 0 | 1 | 1     |
| 1 | 0 | 1     |
| 1 | 1 | 0     |

Table 2. The Results of Bitwise ORing Two Values (A OR B)

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

Table 4. The Results of Bitwise NOTing a Value (A)

| A | ~ A |
|---|-----|
| 0 | 1   |
| 1 | 0   |

Ex: 4 & 8

Ex: 4 | 8

Ex: ~4

## Review: left/right shift

```
unsigned int right = 0xAA00;
```

```
unsigned int rightShift = right >> 8;
```

## Exercise: Compute $a \& b$

Suppose  $a = 7$  and  $b = 4$ .

Give your final result as a hexadecimal number

Exercise: Compute  $a \mid b$

Suppose  $a = 7$  and  $b = 4$ .

Exercise: Compute  $a^b$

Suppose  $a = 7$  and  $b = 4$ .

## Exercise: Compute $\sim a$

Suppose  $a = 7$  and  $b = 4$ .

# Bitops: Code example

```
#include <stdio.h>

int main() {
    unsigned char a = 7;
    unsigned char b = 4;
    unsigned char not = ~a;

    printf("%02X & %02X = %02X\n", a, b, a & b);
    printf("%02X | %02X = %02X\n", a, b, a | b);
    printf("%02X ^ %02X = %02X\n", a, b, a ^ b);
    printf("~%02X = %02X\n", a, not);
    printf("%lu\n", sizeof(unsigned char));

    return 0;
}
```

## Example: Use bit ops to test even/odd

A **mask** is a value that zeroes out all bits except for those that we are interested in.

For example, if we want to extract the rightmost bit, we can use a 0x1 as a mask.

# Demo: Use a mask to test if two signed numbers have opposite signs

What value should we use as a mask?

(Assume 8 bit integers)

# Exercise

```
int value1 = 0;
printf("Enter value 1: ");
scanf(" %d", &value1);

int value2 = 0;
printf("Enter value 2: ");
scanf(" %d", &value2);

unsigned int mask = 0x80000000;
int result1 = mask & value1;
printf("Mask: %08X Value: %08X Result: %08X\n", mask, value1, result1);

int result2 = mask & value2;
printf("Mask: %08X Value: %08X Result: %08X\n", mask, value2, result2);

if (result1 == result2) printf("Your values have the same sign!\n");
else printf("Your values have different signs!\n");
```

# Swapping the lowest two bytes

```
unsigned int a = 0xAABB;

// Goal: swap the two lowest bytes of a.
// Before: 0xAABB
// After: 0xBBAA

// Step 1: Isolate each byte using bitwise AND.
unsigned int upperByte = a & 0xFF00;
unsigned int lowerByte = a & 0x00FF;

// Step 2: Move each byte to its new position.
unsigned int shiftedRight = upperByte >> 8;
unsigned int shiftedLeft = lowerByte << 8;

// Step 3: Combine the two bytes using bitwise OR.
unsigned int swapped = shiftedLeft | shiftedRight;

// Display the intermediate results.
printf("Original:  0x%04X\n", a);
printf("Upper byte: 0x%04X\n", upperByte);
printf("Lower byte:  0x%04X\n", lowerByte);
printf("Shift right: 0x%04X\n", shiftedRight);
printf("Shift left:  0x%04X\n", shiftedLeft);
printf("Swapped:    0x%04X\n", swapped);
```

Original:  
0xAABB  
Upper byte:  
0xAA00  
Lower byte:  
0x00BB  
Shift right:  
0x00AA  
Shift left:  
0xBB00  
Swapped:  
0xBBAA

## Exercise: Swap nibbles

A nibble is 4 bits, or half a byte. Write a program that swaps the nibbles of an unsigned character.

Let  $a = 0xAB$ . Use bitwise operators to compute  $0xBA$

# Exercise: Swap nibbles

```
unsigned char a = 0xAB;

// Step 1: Isolate the upper and lower nibbles.
unsigned char upperNibbleMask = 0xF0;
unsigned char lowerNibbleMask = 0x0F;

unsigned char upperNibble = a & upperNibbleMask;
unsigned char lowerNibble = a & lowerNibbleMask;

// Step 2: Move each nibble to its new position.
unsigned char shiftedRight = upperNibble >> 4;
unsigned char shiftedLeft = lowerNibble << 4;

// Step 3: Combine the shifted nibbles.
unsigned char swapped = shiftedRight | shiftedLeft;

printf("Upper: %02X Lower: %02X\n",
       upperNibble, lowerNibble);

printf("Shifted right: %02X Shifted left: %02X\n",
       shiftedRight, shiftedLeft);

printf("Before: %02X After: %02X\n", a, swapped);
```

## Bit shift: revisited

Recall << shifts to the left

>> shifts to the right

Recall **sign-extension**. How did this work?

How should we “fill in” values when we shift?

# Logical VS arithmetic shifts

## **Logical** shift right

- Fill in with zeros
- Unsigned int behavior

## **Arithmetic** shift right

- Fill in with either 0 or 1 based on the sign bit
  - E.g. prepend with leftmost bit !
- Signed int behavior

Shift left always fills in zero

# Bit shift: example

```
#include <stdio.h>

int main(int argc, char **argv) {
    /* Unsigned integer value: u_val. */
    unsigned int u_val = 0xFF000000;

    /* Signed integer value: s_val. */
    int s_val = 0xFF000000;

    printf("%08X\n", u_val >> 12); // logical right shift
    printf("%08X\n", s_val >> 12); // arithmetic right shift

    return 0;
}
```

# Byte layouts: data type padding

Rule: data is always on a 1/4/8-byte boundary (depending on type size)

```
struct mystery {  
    char c;  
    float q[4];  
};
```

```
sizeof(mystery) = 20 bytes  
mystery.c - 0x7ffcddd3f0e0  
mystery.q[0] - 0x7ffcddd3f0e4  
mystery.q[1] - 0x7ffcddd3f0e8  
mystery.q[2] - 0x7ffcddd3f0ec  
mystery.q[3] - 0x7ffcddd3f0f0
```

# Byte order

The **smallest** addressable unit of memory is a **byte**

If a type is larger than 1 byte, how should we store it?

| Memory Address | Byte Value |
|----------------|------------|
| X:             | AA         |
| X + 1:         | BB         |
| X + 2:         | CC         |
| X + 3:         | DD         |

(a) Big-Endian

| Memory Address | Byte Value |
|----------------|------------|
| X:             | DD         |
| X + 1:         | CC         |
| X + 2:         | BB         |
| X + 3:         | AA         |

(b) Little-Endian

# Bit fields and masking

A **bit field** stores a set of booleans within a single value

A **flag** is a value that can be true or false

Example: Suppose we store 8 flags within a character data type

```
char c = 0x45; // binary is 0100 0101
```

```
// all flags are set to zero except for 3
```

## Example: Dinner options

```
#define WATER 0x01 // mask that corresponds to the least significant bit
#define WINE 0x02 //mask corresponds to the second least significant bit
#define DINNER_ROLL
#define SALAD
#define SOUP
#define MAIN
#define DESSERT
#define COFFEE

// Fill in the remaining masks
```

# Example: Dinner options

What is the bit field corresponding to each of the following options?

| Options                           | Bit Field |
|-----------------------------------|-----------|
| WATER, WINE, SALAD, MAIN, COFFEE  |           |
| WATER, SALAD, SOUP, MAIN, DESSERT |           |
| WINE, MAIN, DESSERT, COFFEE       |           |
| WATER, SALAD, DESSERT             |           |

# Example: Dinner options

What options correspond to the following bit fields?

| Bit Field | Options |
|-----------|---------|
| 0xAE      |         |
| 0x31      |         |
| 0x9B      |         |
| 0x74      |         |

## Example: Dinner options

Write code that sets someone's dinner preferences to WATER, WINE, DESSERT

Write code that tests whether someone choose the MAIN option.

Write code that tests whether someone choose the WINE and DESSERT options.

## Example: File permissions use bit fields

```
alinen@sutekh:~/cs355/os-devel$ ls -l  
-rw-r--r-- 1 alinen alinen 70 Dec 10 2023 README.md
```

Review: What does the above file permissions represent?

What numeric value corresponds to the following permissions (in octal)? `-rw-r--r--`

What numeric value corresponds to the following permissions (in octal)? `-rwxr-x--x`