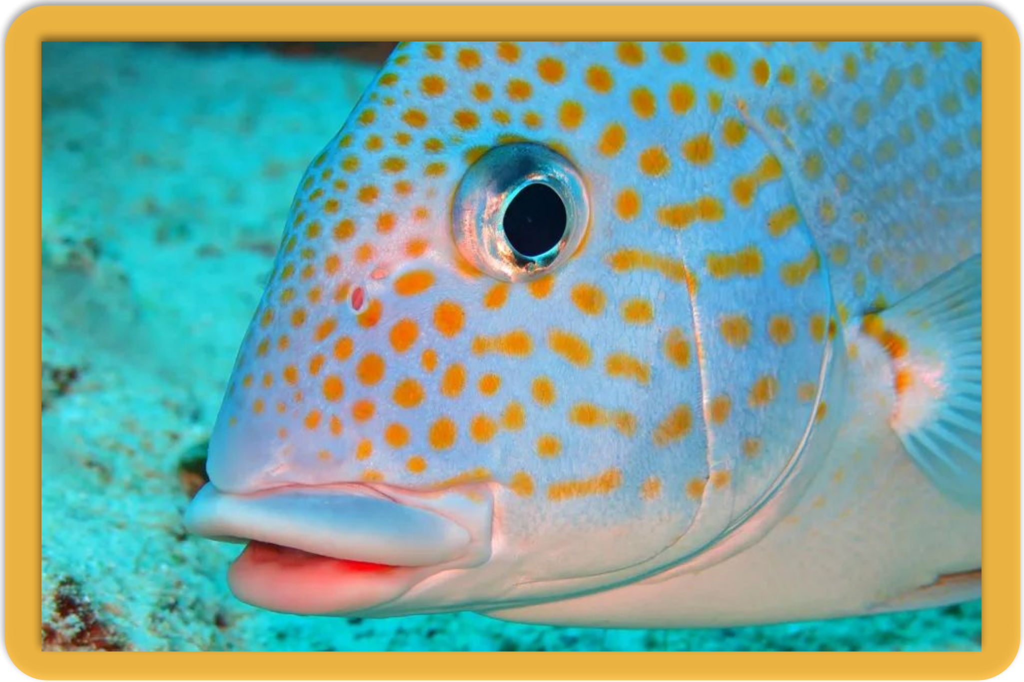
Gear Up:

Data



Project Overview

You will be solving a series of puzzles using your knowledge of data representations.

IMPORTANT: Collaboration

- This project has a different collaboration policy than most projects in CS 33. Please review it!
- As a result, this Gear Up Session will be a little different -- we will not discuss any implementations of any of the code you will write.
- Come ask us questions after if you're unsure.

Topics You Should Review

- Data representation (lecture slides, ch. 2 textbook)
  - Integers (Two’s Complement)
  - Bitwise operations
  - Bit masks and sign extension

Roadmap

`cs0330_install data`

- Read the Worked Example to get an idea of the problems you will be solving
- Read the stencil carefully to know limitations on operators you can use for each puzzle
- Solve each of the 9 puzzles in bits.c
- Use the provided testers to test your solutions as you go

Demos!

Testers

- **btest**
  - In the directory for the project, run `make btest; ./btest`
  - Tests functionality, does not check if you followed the coding rules
- **dpc**
  - In the directory for the project, run `./dpc bits.c`
  - Checks that your code follows the rules for each puzzle, and counts the number of operators
- **driver.pl**
  - In the directory for the project, run `./driver.pl`
  - Combines dpc and btest to check correctness and performance of your solution

Logical Operators

logical **NOT**

A	**NOT A**
T	F
F	T

Logical Operators (C)

logical **NOT**: **!**

A	**!A**
non-zero	00...00
zero	00..01

Bitwise Operators (C)

bitwise **`NOT`**: **`~`**

`A`	`~A`
`11`	`00`
`10`	`01`
`01`	`10`
`00`	`11`

Logical Operators

logical **AND**

A	**B**	**A AND B**
T	T	T
T	F	F
F	T	F
F	F	F

Logical Operators (C)

logical **AND**: **&&**

A	**B**	**A && B**
non-zero	non-zero	1
non-zero	0	0
0	non-zero	0
0	0	0

Bitwise Operators (C)

bitwise **`AND`**: **`&`**

`A`	`B`	`A & B`
1	1	1
1	0	0
0	1	0
0	0	0

Example: `1100 & 1010 = 1000`

Logical Operators

logical **OR**

A	**B**	**A OR B**
T	T	T
T	F	T
F	T	T
F	F	F

Logical Operators (C)

logical **OR**: `||`

A	B	A || B
non-zero	non-zero	1
non-zero	0	1
0	non-zero	1
0	0	0

Bitwise Operations (C)

bitwise **OR**: **|**

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

Example: 1100 | 1010 = 1110

Logical Operators

logical **XOR**

A	**B**	**A XOR B**
T	T	F
T	F	T
F	T	T
F	F	F

Bitwise Operations (C)

bitwise **`XOR`**: **`^`**

`A`	`B`	`A ^ B`
`0`	`0`	`0`
`0`	`1`	`1`
`1`	`0`	`1`
`1`	`1`	`0`

Example 1100 ^ 1010 = 0110

Two's Complement Representation

- The last bit (all the way on the left) is 0 if the number is positive, and 1 if it is negative
- To calculate the decimal representation for a *w*-bit word size (b_i is the bit at index i):

$$-b_{w-1}2^{w-1} + \sum_{i=0}^{w-2} b_i \cdot 2^i$$

Two's Complement Examples

- For a 4-bit word size…
  - 0000
    - $0*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = 0$
  - 1000
    - $-1*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = -8$
  - 1111
    - $-1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = -1$
  - 1010
    - $-1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = -6$

Two's Complement Values

- Values to keep in mind:
  - All `0`'s is `0`
  - All `1`'s is `-1`
  - `0` followed by all `1`'s is the most positive integer
  - `1` followed by all `0`'s is the most negative integer

Bit Shifts (C)

- Shifting the bits in some representation either left or right
- Left Shift: `a << b`
- Right Shift: `a >> b`

Left Shift: << (C)

`a << b`

1. Shift `a` left `b` bits, throwing away leading bits.
   a. How many bits are thrown away?
2. Fill in right bits with 0s.
   a. How many bits are 0s?

Left Shift: Examples

- For a 4-bit word size...
  - `1 << 1 = 2`
    - `0001 << 1 = 0010`
  - `1 << 3 = 8`
    - `0001 << 3 = 1000`
  - `7 << 2 = 12`
    - `0111 << 2 = 1100`
  - `8 << 1 = 0`
    - `1000 << 1 = 0000`

Right Shift: >> (C)

`a >> b`

1. Shift **a** right **b** bits, throwing away trailing bits.
   a. How many bits are thrown away?
2. Logical shift or arithmetic shift?
   a. Logical: Fill in left bits with 0s.
      i. Used for unsigned integers
   b. Arithmetic: Fill in left bits with sign bit.
      i. Used for signed integers

Right Shift: Examples

- For a 5-bit word size, using two's complement...
  - `00001 >> 1`
    - unsigned (logical): `1 >> 1 = 00000 = 0`
    - signed (arithmetic): `1 >> 1 = 00000 = 0`
  - `00111 >> 2`
    - unsigned (logical): `7 >> 2 = 00001 = 1`
    - signed (arithmetic): `7 >> 2 = 00001 = 1`
  - `11001 >> 2`
    - unsigned (logical): `25 >> 2 = 00110 = 6`
    - signed (arithmetic): `-7 >> 2 = 11110 = -2`

Bit Masking

A bit mask is an integer whose binary representation is intended to combine with another value using &, |, or ^ to extract or set a particular bit or set of bits

Bit Masking with &

mask & value1	mask & value2
00000001	00000000

mask	value1	value2
00000001	10011011	10011100

Bit Masking with |

<table>
<tr><th>mask</th><th>value</th><th>mask | value</th></tr>
<tr><td>00101000</td><td>11000101</td><td>11101101</td></tr>
</table>

Bit Masking with ^

mask	value	mask ^ value
11111111	10101010	01010101

Multiplication using <<

- You can multiply a number by a power of 2 by using left-shift (`<<`).
  - Ex: `4*(2^3) = 00000100 << 3 = 00100000 = 32`

Dividing using >>

You can use >> to divide an integer by a power of 2

- Positive integers
  - Ex: `48/(2^4) = 00110000 >> 4 = 00000011 = 3`
  - Ex: `48/(2^5) = 00110000 >> 5 = 00000001 = 1`
    - Throwing away trailing bits rounds down
- Negative integers
  - Ex: `-48/(2^4) = 11010000 >> 4 = 11111101 = -3`
  - Ex: `-48/(2^5) = 11010000 >> 5 = 11111110 = -2`
    - WRONG! Just using >> on a negative number rounds away from zero (bad)

Correct Rounding for Negative Ints

- Problem: we want to increment our answer only in the case where there’s rounding away from 0

`-48 / (2^5) = 11010000 >> 5 = 11111111`

  - **Only** if there is a one in the trailing bits that will be cut off (ie: the answer will be rounded), then the leading bits need to be incremented by 1.
  - How tho?? Hint: check the lecture slides ;)

Written Question

- One written question about floating points,
  - 2 parts
- Each part will be worth 10 points
  - the project is 120 points total
- Most of the needed information is on the lecture slide
  - Apply the formulas on the slides!

Tips

- Try to follow the process used in the worked example.
- Understanding the lectures on bit manipulation and two’s complement will be very important for this project
- Making a bit mask using the sign bit will be very useful (**x >> 31**)
- Come to TA hours if you have questions! TAs will try to help without giving away the answer

So Why Am I Doing This?

- You will gain a better understanding of bit-level representation of data.
- You will gain a better understanding of how the computer computes functions like negating a number or checking if numbers are equal.
- You will understand the various bit-level operations

Questions?