ECE 120 Midterm 1B

HKN Review Session

Time (self-scheduled): Between Wednesday, September 27 and Friday, September 29, 2017
Location: 57 Grainger Engineering Library (in the basement on the east side)
Schedule Your Exam: cbtf.engr.illinois.edu (up to two weeks in advance)
What to bring: iCard, pens/pencils (They provide the scratch paper)

Overview of Review

- What to remember
- Abstraction
- Binary
- Operators
- Bitmasks
- C programming
- Practice Exam

Levels of Abstraction

1. The Problem - an ambiguous outline of the task at hand
2. The Algorithm - an unambiguous approach to solving said problem
3. Programming Language - means in which humans communicated their algorithm to the computer, ideally, without ambiguity.
4. Instruction State Architecture (ISA) - The basic set of operations available at the hardware level of a microprocessor. Eg. x86, LC-3, and MIPS
5. Microarchitecture - High level chip architecture/organization for performance
6. Elements of Microarchitecture - Implements various elements as Logic Circuits
7. Device - Actual materials and construction of parts composing logic circuits

Binary Representation

- Unsigned - k bits can represent [0, 2^k) values
  - Can only represent non-negative integers
  - Overflow Condition: If the most significant bit has a carry out
  - Zero Extended - pad the front with zeros when moving to larger data type
  - (10101) = (16+4+1) = 21
- Signed Magnitude - (- 2^(k-1), 2^(k-1)) values
  - First bit determines sign of number (1 = negative, 0 = positive)
  - (10101) = (-1) x (4+1) = -5

Binary Representation - Part II

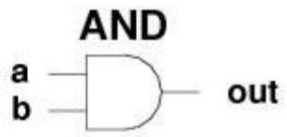
- 2’s Complement - k bits represents [-2^(k-1), 2^(k-1) - 1]
  - Sign Extended - pad the front of the number with the sign bit when moving to larger data type
  - Overflow Condition: if adding numbers of the same sign yields a result of the opposite sign
  - If signed bit is 0, magnitude is same as if the number were treated as unsigned
  - If signed bit is 1, to determine the magnitude, bitwise NOT the bits, and add 1 before finding magnitude
  - 2’s complement provides a greater range of values and cleaner binary arithmetic operations
    - ie. the same logic circuit used to add binary unsigned and 2’s complement numbers

iEEE 754 Floating Point Representation

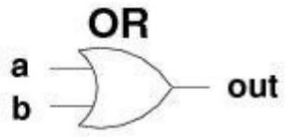
- Great for approximations & expressing very large/very small decimal values
- 1st bit is sign bit, 0 for positive and 1 for negative
- 2nd through 9th bit is exponent
  - Special Cases: When exponent is 0 or 255, then denormalized or NAN/inf forms respectively.
  - Denormalized format has normal sign, and magnitude: 0.mantissa * 2^(-126)
  - NAN/inf form is NAN when mantissa is non-zero, and infinity (inf) when mantissa is all zeros
- 10th through 32nd bit is 23 bit mantissa
- Normal Form has magnitude of 1.mantissa * 2 ^ (exponent - 127)
- (-1)^sign * 1.fraction * 2^(exponent - 127)

Operators

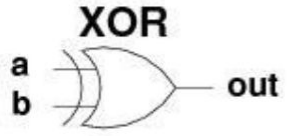
- AND
  - Returns 1 if both inputs are 1
- OR
  - Returns 1 if any of the inputs are 1
- XOR
  - Returns 1 if either a or b is 1, not both
- NOT
  - Returns opposite of input (0->1 , vice versa)



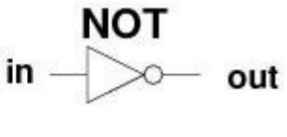
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1



a	b	out
0	0	0
0	1	1
1	0	1
1	1	1



a	b	out
0	0	0
0	1	1
1	0	1
1	1	0



in	out
0	1
1	0

Bitmasks

- Great for looking at only certain bits
- A bitmask using AND as the bitwise operator will selectively mask bits to 0
- A bitmask using OR as the bitwise operator will selectively mask bits to 1

C programming

- Variables
  - Double, Long (8 bytes)
  - Int, Float (4 bytes)
  - Short (2 bytes)
  - Char (1 byte)

```c
#include <stdio.h>
#define PI = 3.1415926;

int main(){

    int r = 12;

    float area;

    area = (PI * r * r);

    return 0;
}

```

Format Specifiers and Escape Sequences

Format Specifiers:	Escape Sequences:
%c char single character %d (%i) int signed integer %e (%E) float or double exponential format %f float or double signed decimal %g (%G) float or double use %f or %e as required %o int unsigned octal value %p pointer address stored in pointer %s array of char sequence of characters %u int unsigned decimal %x (%X) int unsigned hex value	\n - new line character \b - backspace character \t - horizontal tab \’ - allows for storing and printing of ‘ ASCII value in string \\ - allows for storing and printing of \ ASCII value in string \” - allows for storing and printing of “ ASCII value in string \? - allows for storing and printing of ? ASCII value in string Ie. char x = ‘\’’; (those are 2 single quotes) will store the data byte associated with ‘ into x

C - operators

- Order of precedence
  - * , / , % , then + , - (Note, Modular Arithmetic (%) is not defined for floating point numbers)
- Assignment operator
  - = (takes an lvalue on the left side and an rvalue on the right side to set the lvalue to the rvalue)
    - IE, you can't do 5 = x; as 5 is an rvalue (you can't assign a value to 5!)
- Relational
  - ==, !=, <, >, <=, >=
- Bitwise
  - &, |, ~, ^ (AND, OR, NOT, XOR)
- Logical
  - !, &&, || (NOT, AND, OR)

C operators - bitwise examples

```c
int a = 12;
int b = 20;
int c;

c = (a|b);
```

```c
int a = 13;
int b = 6;
int c = 17;
int d;

d = (((a|b)&c) | b);
```

```c
int a = 7;
int b = 16;
int c = 21;
int d = 12;
int e;

e = (((a^b)&c) & ((d^a)|b));
```

Basic Input / Output

```c
float i = 2.15;
int j;

scanf("%d", &j);

printf("%d and %f", j, i);
```

Conditional Constructs

```c
int a = 4;

if (a < 10){
    printf("a is less than 10");
}
```

```c
int a = 4;
int b = 8;

if (a < 10){
    printf("a is less than 10");
}else if (b < 20){
    printf("b is less than 20");
}else{
    printf("b is greater than 20");
}
```

Conditional Constructs - Example

```c
int a = 20;
int b = 13;
int c = 6;

if ( (a-b > 10) || ((c*a)%10 == 0)){
    if(b+c < a){
        printf("a = %d", a);
    }else{
        printf("Error");
    }
}else{
    printf("b = %d", b);
}
```

Iterative Constructs

```c
int i = 0;

do {
    /* do things */
} while(i < 10);
```

```c
int i;

for(i = 0; i<=10; i++){
    printf("%d", i);
}
```

```c
int i = 0;

while (i < 10){
    /* do things*/
}
```

Iterative - Example

```
int i;

for(i = 0; i<=10; i++){
    if (i%3 == 0){
        printf("%d\n", i);
    }
}
```

Convert the decimal value -47 to 7-bit two's complement.

1010001

This question is complete and cannot be answered again.

Zero-extend the 6-bit unsigned number 111001 to a 8-bit unsigned number.

00111001

This question is complete and cannot be answered again.

$$\begin{array}{r} 001110 \\ +100010 \end{array}$$

Solve the above 6-bit unsigned addition problem

110000

- ○ Overflow
- ◉ No Overflow

This question is complete and cannot be answered again.

What is the decimal value of the largest **positive two's complement** number that can be represented using 10 bits?

511

This question is complete and cannot be answered again.

```
  111001
+ 011000
```

Solve the above 6-bit two's complement addition problem

010001

- ( ) Overflow
- (•) No Overflow

This question is complete and cannot be answered again.

Convert the character $ to a 7-bit ASCII value.

A copy of the table of ASCII characters is available for your reference.

0100100

Convert the 2-digit hexdecimal value 1F to 8-bit binary.

00011111

Compute the bitwise NOT for the following 7-bit binary string.

0010101

1101010

This question is complete and cannot be answered again.

Compute the bitwise OR for the following 8-bit binary strings.

01110011

00110101

01110111

This question is complete and cannot be answered again.

Convert the single-precision IEEE 754 floating point value 0 10000000 11100000000000000000000 to decimal.

3.75

This question is complete and cannot be answered again.

Midterm 1B practice Exam

Select a logical operation and enter a bitmask that convert the following input bit strings to the corresponding output bit strings.

$01000001 \rightarrow 01000000$

$01000110 \rightarrow 01000000$

$10111111 \rightarrow 10110000$

Logical operation

- (•) AND
- ( ) OR

Bitmask (8 bits) 11110000

$$\begin{array}{r} 011110 \\ -011111 \end{array}$$

Solve the above 6-bit two's complement subtraction problem

111111

- ○ Overflow
- ◉ No Overflow

Convert the 5-bit two's complement value 10000 to decimal.

-16

Consider the following fragment of C code. The variable `score` has type `int`.

```
if (score<=106) {
    printf("F");
} else if (score<=124) {
    printf("D");
    if (score<113) { printf("-"); }
    if (score>121) { printf("+"); }
} else if (score<=142) {
    printf("C");
    if (score<131) { printf("-"); }
    if (score>139) { printf("+"); }
} else if (score<=160) {
    printf("B");
    if (score<149) { printf("-"); }
    if (score>157) { printf("+"); }
} else if (score<=180) {
    printf("A");
    if (score<167) { printf("-"); }
    if (score>175) { printf("+"); }
}
```

What is the range of values of `score` that makes the fragment of code print out exactly the following?

```
D+
```

122 ≤ score ≤ 124

Consider the following fragment of C code.

```
for (i=1; i<3; i=i+1)
{
    /* This is the loop body. */
}
```

How many times does the loop body execute?

2

Consider the following fragment of C code. The variables x, y and z all have type `int`. Section 1.5.4 of Lumetta's notes explains the meaning of C operators.

```
x = 54;
y = 3;
z = x & y;
```

Enter the decimal value of the variable z after the code has executed.

2

The C program below prints out (in some order) the 8-bit 2's complement representation of a decimal number. Note that the variable `number` has type `int8_t`, which is 8-bit 2's complement.

```
#include <stdio.h>
#include <stdint.h>

int main()
{
    int8_t number;
    int i;

    scanf("%d", &number);

    for (i=0; i<8; i=i+1) {

        /* The bitmask '& 0x01' masks out all but the rightmost bit */
        if ((number & 0x01) == 0) {
            printf("0");
        } else {
            printf("1");
        }
        number = number >> 1;

    }

    return 0;
}
```

What decimal value (between -128 and 127 inclusive) must the user input at the `scanf` to print the following output to screen?

```
10111111
```

-3

Consider the following fragment of C code. The variables `x` and `z` all have type `int`. Section 1.5.4 of Lumetta's notes explains the meaning of C operators.

```
x = 29;
z = ~x;
```

Enter the decimal value of the variable `z` after the code has executed.

-30

Consider the following fragment of C code.

```
int a = 2;
float b = 2;
int c = a-b;
printf("FORMATSTRING", a, b, c);
```

Enter the text that should replace `FORMATSTRING` so that the fragment of code produces the following output. You may use the format specifiers `%d` and `%f`, but not others.

```
(2)-(2.000000)=0
```

(%d)-(%f)=%d

Convert the decimal value -0.6 to single-precision IEEE 754 floating point representation. Enter the first 16 bits only. You may use spaces to separate groups of bits. The last 16 bits are given to you as shown below.

First 16 bits: 1 01111110 0011001

Last 16 bits: 1001100110011001

Indicate whether the full 32-bit representation is exact or approximate representation of -0.6.

- ○ Exact
- ◉ Approximate