ECE 120 Midterm 1

HKN Review Session

Exam Time: Tuesday, February 14 (7:00-8:30pm)

Logistics

Exam: Tuesday, February 14, 7pm-8:30pm

Conflict exam: 5pm-6:30pm
Location: Check Compass for room assignment

UA Review Session: Sunday, February 12, 2pm-4pm

Overview of Review Session

- Abstraction
- Binary Types & Hexadecimal & Overflow
- Floating Point
- Boolean Operators
- C Programming
- FALL 2015 EXAM QUESTIONS
Abstraction/Levels of Transformation

- Abstraction - the means to simplify events without going into heavy specifics, reducing information to the essentials -> productivity enhancer
- Levels of Transformation (itself an example of abstraction !)
  - Problem Statement
  - Algorithm
  - Program (C)
  - Instruction Set Architecture (MIPS, LC-3 assembly language)
  - Microarchitecture (combinational/sequential logic circuits)
  - Logic gates (NOT AND OR)
  - Devices (CMOS)
Binary Types

- **Unsigned**
  1. Can only represent *nonnegative* integers
  - $K =$ number of bits
  - Total unique representations $\rightarrow 2^K$
  - Range $\rightarrow 0$ to $(2^K-1)$
    - e.g. $(10011)_2 \rightarrow (16+2+1) = 19_{10}$

  Decimal - Binary conversion: represent both as a sum of 2’s powerful numbers.

- **Signed - Magnitude (rarely used)**
  1. First bit determines if positive or negative $\rightarrow 1 =$ negative, 0 = positive
  - Rest of bits determine magnitude
  - Range $\rightarrow -(2^{(K-1)} - 1)$ to $(2^{(K-1)} - 1)$
    - e.g. $(10011)_2 \rightarrow (-1) \times (2+1) = -3_{10}$
Binary Types *

- **2’s complement**
  - Positive numbers lead with “0”, negative numbers lead with “1”
    - K bits → can represent $2^k$ total numbers, half being positive and half being negative
  - Can represent positive numbers from range $-(2^{(k-1)})$ to $(2^{(k-1)} - 1)$
  - **Positive** numbers have the same representation as unsigned types (with the MSB being zero); for **negative** numbers, do the following two steps to find their 2’s complement representations from unsigned representations:
    - e.g. $(-37)_{10}$
      - **Find the corresponding positive** 2’s complement value first (sign bit!)
      - **FLIP ALL BITS & ADD 1**

Positional Weighting method (IN HW2!): quick way from 2’s complement to decimal!
- the kth bit of the number has the “weight” of $2^{(k-1)}$
- except for the leftmost bit (MSB) which has the “weight” of $-2^{(k-1)}$
Overflow in Operations

- 2 primary operations: addition and subtraction (essentially +)
- Checking for Overflow
  - Unsigned operations
    - There is a nonzero carry bit (bit carries out of bit range)
  - 2’s Complement operations
    - Result has wrong sign if
      - 2 positive numbers sum to negative number
      - 2 negative numbers sum to positive number
    - NOTE: in 2’s complement, a positive and negative number added never results in overflow
    - Quick Check- For MSB, does carry in bit = carry out bit (i.e. $C_n = C_{n-1}$)?
      - If not, overflow has occurred
Floating Point

- Use IEEE 754 standard (32 total bits)
  - 1 sign bit
  - 8 exponent bits
  - 23 mantissa bits
- Increased precision
  => decreased range
- Conversion from floating point to decimal
- Conversion from decimal to floating point
Floating Point (cont.)

- Special Cases
  - Denormalized representation
    - Exponent = 0
      - Mantissa takes any value
      - Formula: \((-1)^e \times 0.\text{Mantissa} \times 2^{-126}\)
    - Exponent is all 1s
      - Mantissa = 0
  - \((-1)^e \times \infty\)
    - Mantissa not equal to 0
  - NaN

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Mantissa [Fraction]</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ≤ exponent ≤ 254</td>
<td>All values</td>
<td>((-1)^{\text{sign}} \times (1.\text{mantissa}) \times 2^{\text{exponent}})</td>
</tr>
<tr>
<td>(Normalized)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exponent = 0</td>
<td>All values</td>
<td>((-1)^{\text{sign}} \times (0.\text{mantissa}) \times 2^{-126})</td>
</tr>
<tr>
<td>(Denormalized)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>exponent = 255</td>
<td>0</td>
<td>((-1)^{\text{sign}} \infty)</td>
</tr>
<tr>
<td>(Overflow)</td>
<td>Non-zero</td>
<td>NaN (Not a Number)</td>
</tr>
</tbody>
</table>
Boolean Operators *

- NOT
- AND, NAND - AND allows masking of bits

Mask: 00001111  Value: 01011100  Result: 00001100

- XOR, XNOR
  1  A XOR B = A \( \text{NOT} \) B + \( \text{NOT} \) A \( \text{B} \)

- OR, NOR  
  1  Order of precedence:
     - (), NOT, AND, OR
     - AND, NOT, and OR are logically complete
Hexadecimal *

- Base 16, Uses 0-9 and A-F
- Takes groups of 4 bits and represents them as symbols
  1. Ex: 0011 1101 0110 1110 → 3 D 6 E

- To go from hex to binary, write out each hex value into 4 bit binary
  1. Ex: 4E7F → 0100 1110 0111 1111

- Shortens binary representation by a factor of 4

Octal Notation:
Base 8, Uses 0-7
Takes groups of 3 bits and represents them as numbers from 0-7

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>
C Programming

- Basic Characteristics
  1. High level/independent (of ISA), procedural, expressive

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

int main() {
    int r = 10;

    float area;
    area = pi * r * r;
    return 0;
}
```
- Variables in C
  - Int, double, float, char
    1. Note that result is truncated during integer division!

```c
int r = 10;
```

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
</table>

---
Operators

- Order of precedence: *, /, % and then +, -
  - Assignment operator: =
  - Relational: ==, !=, >, <, >=, <=
  - Bitwise: &, |, ~, ^ (AND, OR, NOT, XOR)
  - Logical: &&, ||
- Basic I/O

```c
int a = 1;
float b = 0.1;
printf("a=%d, b=%f\n", a, b);
```
Conditional Constructs

```c
int a = 5;
if (a < 10) {
    printf("a is less than 10\n");
}
```

```c
int a = 5;
int b = 8;
if (a < 10) {
    printf("a < 10\n");
} else if (b < 10) {
    printf("b < 10\n");
} else {
    printf("b >= 10\n");
}
```
Iterative Constructs

```java
int i = 0;
while (i < n) {
    /* Do something. */
    i ++;
}
```

Note: do-while loop will be able to get into the loop for at least once

```java
int i;
for (i = 0; i < n; i ++) {
    /* Do something. */
}
```

```java
int i = 0;
do {
    /* Do something. */
    i ++;
} while (i < n);
```
Flow Chart Components
Cheat Sheet: Recommendations

- Common powers of 2
- 2’s Complement
  1. Procedure of converting between decimal & binary types
     Representable range with K bits
- Floating Point
  1. Formula for general case ○ Special cases
- Overflow Conditions (both unsigned and 2’s complement)
- Harder boolean operators
  1. XNOR, XOR, NAND, NOR
- Basic C syntax
General Advice

- Use your Cheat Sheet! Don’t memorize
- Read the directions carefully!!!!
- Don’t be afraid to ask questions
- Relax and trust what you’ve learned :)}