CSCI3240 Exam 1 Study Guide

Exam 1 Question Format

1. Multiple choice
2. Fill in the blanks
3. True or False

Chapter 2: Practice Problems

I. Assume we are running code on a 8-bit machine using two's complement arithmetic for signed integers. A "short" integer is encoded using 4 bits. Fill in the empty boxes in the table below. The following definitions are used in the table:

```
short sy = -3;
int y = sy;
int x = -17;
unsigned ux = x
```

Note: You need not fill in entries marked with "–"

Only accepted format for binary representation (use 8-bits): 00000000, 11111111

Only accepted format for hexadecimal representation: 4D, EF, FF

Expression	Decimal	Binary	Hexadecimal
Zero	0		
-	-3		
-		00110010	
ux			
y			
x >> 2			
Tmax			
Tmax+Tmin			
Tmin - 1			

II. Assume we are running code on a 10-bit machine using two's complement arithmetic for signed integers. Fill in the empty boxes in the table below. The following definitions are used in the table:

int y = -9;

unsigned z = y;

Note: You need not fill in entries marked with "–"

Only accepted format for binary representation (use 10-bits): 0000000000, 1111111111

Only accepted format for hexadecimal representation: 2FD, 1EF, 3FF

Expression	Decimal	Binary	Hexadecimal
Zero	0		
-	-5		
–		0000010010	
y			
z			
y-z			
-Tmax			
-Tmin			
Tmax +1			

III. Integer puzzles

Check if the statements are always true?

Initialization

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

```
1. x < 0          => ((x*2) < 0)
2. ux >= 0
3. x & 7 == 7     => (x<<30) < 0
4. ux > -1
5. x > y          => -x < -y
6. x * x >= 0
7. x > 0 && y > 0 => x + y > 0
8. x >= 0         => -x <= 0
9. x <= 0         => -x >= 0
10.(x|-x)>>31 == -1
11.ux >> 3 == ux/8
12.x >> 3 == x/8
13.x & (x-1) != 0
```

IV. What is the output of the following code?

Assume that int is 32 bits, short is 16 bits, and the representation is two's complement.

```
unsigned int x = 0xDEADBEEF;
unsigned short y = 0xFFFF;
signed int z =-1;
if (x > (signed short) y)
  printf("Hello");
if (x > z)
  printf("World");
```

(a) Prints nothing.
(b) Prints "Hello"
(c) Prints "World"
(d) Prints "HelloWorld"

V. After executing the following code, which of the variables are equal to 0?

```
unsigned int a = 0xffffffff;
unsigned int b = 1;
unsigned int c = a + b;
unsigned long d = a + b;
unsigned long e = (unsigned long)a + b;
(Assume ints are 32 bits wide and longs are 64 bits
wide.)
```

(a) None of them

(b) c

(c) c and d

(d) c, d, and e

VI. Floating point representation

Consider a 12-bit variant of the IEEE floating point format as follows:

- Sign bit
- 5-bit exponent with a bias of 15.
- 6-bit significand

All of the rules for IEEE 754 Standard apply.

Fill in the numeric value represented by the following bit patterns. You **must** write your number in decimal form (e.g. 0.0146485375, -0.0146485375).

Bit Pattern	Numerical Value
010011101110	
111011101011	
100101001111	
001010111010	

VII. Floating points puzzles

- **For each of the following C expressions, either:**
  - Argue that it is true for all argument values
  - Explain why not true

```
int x = …;
float f = …;
double d = …;
```

Assume neither `d` nor `f` is NaN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (double)(float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0 ⇒ ((d*2) < 0.0)`
- `d > f ⇒ -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

Chapter 3: Practice Problems

VIII. You are given the following C code to compute integer absolute value:

```
int abs(int x)
{
      return x < 0 ? -x : x;
}
```

You've concerned, however, that mispredicted branches cause your machine to run slowly. So, knowing that your machine uses a two's complement representation, you try the following (recall that sizeof(int) returns the number of bytes in an int):

```
int opt_abs(int x)
{
int mask = x >> (sizeof(int)*8-1);
int comp = x ^ mask;
return comp;
}
```

A. What bit pattern does mask have, as a function of x?

B. What numerical value does mask have, as a function of x?

C. For what values of x do functions abs and opt abs return identical results?

D. For the cases where they produce different results, how are the two results related?

E. Show that with the addition of just one single arithmetic operation (any C operation is allowed) that you can fix opt abs. Show your modifications on the original code. *(You can just provide the line that you will add).*

F. Are there any values of x such that **abs** return a value that is ***not*** greater than 0? Which value(s)?

IX. Consider the following C functions and assembly code

```
long functionA(long a){
  return a * 30;
}

long functionB(long a){
  return a * 34;
}

long functionC(long a){
  return a * 16;
}

long functionD(long a){
  return a * 18;
}

long functionD(long a){
  return a * 36;
}
```

```
Assembly code:
movq %rdi, %rax
salq $3, %rax
addq %rdi, %rax
addq %rax, %rax
retq
```

Which of the functions compiled into the assembly code shown?

X. Consider the following C functions and assembly code

Assume that long is 64 bits, int is 32 bits, short is 16 bits, and the representation is two's complement.

Assembly Code:

```
imulq  %rsi, %rdi
```

```
imulq  %rdx, %rsi
addq   %rsi, %rdi
leaq   (%rdi,%rdi,2), %rax
salq   $3, %rax
ret
```

```
long functionA(long a, long b, long c){
long d = a*b;
long e = b*c;
return 18 * (d+e);
}
```

```
long functionB(long a, long b, long c){
long d = a*b;
long e = b*c;
return 24 * (d*e);
}
```

```
long functionC(long a, long b, long c){
long d = a*b;
long e = b*c;
return 24 * (d+e);
}
```

```
long functionD(long a, long b, long c){
long d = a*b;
long e = b*c;
return 32 * (d*e);
}
```

Which of the functions compiled into the assembly code shown?

XI. What is the C equivalent of `mov 0x44(%rax,%rcx,8), %rdx`

```
(a)  rdx = rax + rcx + 8 + 44
(b) *(rax + rcx + 8 + 10) = rdx
(c) rdx = *(rax + rcx*8 + 0x44)
(d) rdx = *(rax + rcx + 8 + 0x44)
```

XII. Reconstruct the following C code for this recursive function by looking at the assembly code. Fill in the blanks:

```
unsigned myfunction2(unsigned n)

{
    if (_____________) return 1;
    else {
        return 1 + myfunction2(_____________);
    }
}
```

```
myfunction2:
    testq    %rdi, %rdi
    jne  .L9
    movq $1, %rax
    ret
.L9:
    subq $8, %rsp
    shrq $2, %rdi
    call myfunction2
    addq $1, %rax
    addq $8, %rsp
    ret
```