Login: cs30x_______  Name _________________________
Student ID ____________________  Signature_______________________

By filling in the above and signing my name, I confirm I will complete this exam with the utmost integrity and in accordance with the Policy on Integrity of Scholarship.

**CSE 30**  
**Fall 2013**  
**Final Exam**

1. **Number Systems / Right-Left Rule**  
   ________________  (22 points)

2. **Binary Addition/Condition Code Bits/Overflow Detection**  
   ________________  (12 points)

3. **Branching**  
   ________________  (21 points)

4. **Bit Operations**  
   ________________  (13 points)

5. **Recursion/SPARC Assembly**  
   ________________  (10 points)

6. **Local Variables, The Stack, Return Values**  
   ________________  (20 points)

7. **More Recursive Subroutines**  
   ________________  (8 points)

8. **Floating Point**  
   ________________  (12 points)

9. **Machine Instructions**  
   ________________  (21 points)

10. **Linkage, Scope, Lifetime, Data**  
    ________________  (34 points)

11. **Load/Store/Memory**  
    ________________  (11 points)

12. **Miscellaneous**  
    ________________  (23 points)

**SubTotal**  
______________  (207 points)

**Extra Credit (~5%)**  
______________  (10 points)

**Total**  
______________

This exam is to be taken by yourself with closed books, closed notes, no electronic devices. You are allowed both sides of an 8.5"x11" sheet of paper handwritten by you.
1. Number Systems

Convert 0xFB8E (2’s complement, 16-bit word) to the following. (6 points)

**binary**_____________________________________ (straight base conversion to binary)

**octal**_______________________________________ (straight base conversion)

**decimal**____________________________________ (convert to signed decimal)

Convert 274 to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

**sign-magnitude**

**1’s complement**

**2’s complement**

Convert -695 to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

**sign-magnitude**

**1’s complement**

**2’s complement**

**Rt-Lt Rule**

Using the C Rt-Lt Rule, define a variable named fubar that is a multi-dimensional array of 17 rows and 35 columns where each element is a pointer to a function that takes a single argument of type pointer to double and returns a pointer to an array of 23 elements where each element is of type pointer to struct foobar. (7 points)

2. Binary Addition/Condition Code Bits/Overflow Detection

Indicate what the condition code bits are when adding the following 8-bit 2’s complement numbers. (12 points)

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01010110       +00111011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10110100       +01001100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10011101       +10101001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Branching (21 points)
Translate the C code below into the equivalent unoptimized SPARC Assembly code. Just perform a direct translation – no optimizations. Use the local register mappings for the variables in assembly as specified. Remember to implement short-circuiting logic for the logical OR.

```
C
/* Assume variables a and b have been properly declared as ints. */

if ( (a <= b) || (b > 99) )
{
    do
    {
        a = a * b;
    } while ( a < 55 );

    ++b;
}
else
{
    b = b - 22;
}

SPARC ASSEMBLY
! a is mapped to %l3
! b is mapped to %l6
```
4. Bit Operations

What is the value of %l0 after each statement is executed? **Express your answers as 8 hexadecimal digits.**

(All 32 bits. Be sure to specify any leading or trailing zeros.)

set 0xFEEDBABE, %l0
set 0x86715309, %l1
and %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFEEDBABE, %l0
sra %l0, 7, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFEEDBABE, %l0
sll %l0, 7, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFEEDBABE, %l0
set 0x????????, %l1
xor %l0, %l1, %l0

! Value in %l0 is now Ox86715309

Value set in %l1 must be this bit pattern _______________________________________ (3 points)

set 0xFEEDBABE, %l0
set 0x86715309, %l1
or %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFEEDBABE, %l0
srl %l0, 6, %l0

Value in %l0 is _______________________________________ (2 points)
5. Recursion/SPARC Assembly
Given main.s and fubar.s, what gets printed when executed? Yes … Draw stack frames! (10 points)

```assembly
.global main                    /* main.s */
.section ".rodata"
.align 4
.word 0x30003300
.code: .word 0x33736447, 0x72657074, 0x663D4E37, 0x2B20422B, 0x66754372
.word 0x662F7522, 0x43537300, 0x00000030
.main:
save %sp, -92 & -8, %sp
.set code, %o0
.mov 27, %o1
.call fubar
.nop
.ret
.restore

.global fubar                   /* fubar.s */
.section ".rodata"
.fmt: .asciz "%c"
.section ".text"
.fubar:
save %sp, -(92 + 1) & -8, %sp
.cmp %i0, %g0
.be end
.nop
.ldub [%i0 + %i1], %l0
.cmp %l0, %g0
.be end
.nop
.dec %l0
.stb %l0, [%fp - 1]
.sub %i1, 3, %o1
.mov %i0, %o0
.call fubar
.nop
.set fmt, %o0
.ldub [%fp - 1], %o1
.call printf
.nop
.end:
.ret
.restore
```

What gets printed? _____________________________
6. Local Variables, The Stack, and Return Values
Here is a C function that allocates a few local variables, performs some assignments and returns a value. Don’t worry about any local variables not being initialized before being used. Just do a direct translation. **Draw lines.**

```c
int fubar(int a, char b) {
    int *local_stack_var1;
    int   local_stack_var2;
    struct foo
    { int s1[3];
      char s2;
      int s3;
      char s4[8];
    } local_stack_var3;

    local_stack_var3.s4[3] = 'A'; /* 1 */
    local_stack_var2 = b++ + local_stack_var3.s1[2]; /* 2 */
    local_stack_var3.s2 = local_stack_var3.s4[1] - a; /* 3 */
    return ( ++*local_stack_var1 + 420 ); /* 4 */
}
```

Write the equivalent **full unoptimized** SPARC assembly language module to perform the equivalent. **You must allocate all local variables on the stack.** No short cuts. Treat each statement independently. (20 points)
What is the output of the following program? (8 pts)

```c
#include <stdio.h>

int AAA( int x1 )
{
    int r1;
    printf( "x1 = %d\n", x1 );
    if ( x1 <= 1 )
    {
        return 1;
    }
    else
    {
        r1 = BBB( x1 - 1 ) + x1;
        printf( "r1 = %d\n", r1 );
        return r1;
    }
}

int BBB( int x2 )
{
    int r2;
    printf( "x2 = %d\n", x2 );
    if ( x2 <= 2 )
    {
        return 2;
    }
    else
    {
        r2 = AAA( x2 - 2 ) + x2;
        printf( "r2 = %d\n", r2 );
        return r2;
    }
}

int main( int argc, char *argv[] )
{
    printf( "%d\n", BBB( 6 ) );
    return 0;
}
```

Put output here
8. Floating Point

Convert 139.75_{10} (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary fixed-point</td>
<td>______________</td>
</tr>
<tr>
<td>IEEE floating-point</td>
<td>______________</td>
</tr>
</tbody>
</table>

Convert 0xC335E000 (single-precision IEEE floating-point representation) to fixed-point decimal.

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-point decimal</td>
<td>______________</td>
</tr>
</tbody>
</table>

9. Machine Instructions

Translate the following instructions into SPARC machine code. Use hexadecimal values for your answers. If an instruction is a branch, specify the + or - number of instructions away for the target (vs. a Label).

|xorcc %o2, %l5, %i1        | ________________________ |  (5 points)
| sth %i4, [%l3 + 12]       | ________________________ |  (5 points)

Translate the following SPARC machine code instructions into SPARC assembly instructions.

|xorcc %o2, %l5, %i1        | ________________________ |  (5 points)
| sth %i4, [%l3 + 12]       | ________________________ |  (5 points)

If an odd-ball computer had 421MB of memory, how many bits would be needed in an address register to address any byte in this system?

_____ (1 point)
10. Linkage, Scope, Lifetime, Data

For the following program fragment, specify in which C runtime area/segment each symbol will be allocated or pointing: (34 points — 1 point each)

```c
int a = -17; ______________ (a)
static int b; ______________ (b)
static int c = 37; ______________ (c)
int d; ______________ (d)

int foo( int e ) { ____________ (e) ____________ (foo)
    static double f = 42.24; ______________ (f)
    static int *g;
    g = (int *) malloc( e ); ______________ (where g is pointing)
    int (*h)(int) = foo; ____________ (h) ____________ (where h is pointing)
    double i = f;
    ____________ (i)

    ... }
```

Fill in the letter corresponding to the correct **scoping/visibility** for each of the variables:

A) Global across all modules/functions linked with this source file (global scope).
B) Global just to this source file (file scope).
C) Local to function foo() (local/block scope).

a _______ a _______
b _______ b _______
c _______ c _______
d _______ d _______
e _______ e _______
f _______ f _______
g _______ g _______
h _______ h _______
i _______ i _______
foo _______

Fill in the letter corresponding to the correct **lifetime** for each of the variables:

A) Exists from the time the program is loaded to the point when the program terminates.
B) Exists from the time function foo() is called to the point when foo() returns.

If the function foo() above is called 10 times, indicate how many times will f be initialized? __________
If the function foo() above is called 10 times, indicate how many times will i be initialized? __________
11. Load/Store/Memory
Specify the all 8 hex values requested after those lines have been fully executed. (11 points)
(All 32 bits. Be sure to specify any leading or trailing zeros.)

.globaI main

.section ".data"
fmt: .asciz "0x%08X\n" ! prints value as hex 0XXXXXXXX

.c: .byte 0xDD
align 2

s: .half 0xCAFE
.align 4

i1: .word 0x13579BDF
i2: .word 0x13579BDF
i3: .word 0x13579BDF
x: .word 0x87650000

.section ".text"
main:
save %sp, -96, %sp
set x, %l0
set s, %l1
ldsh [%l1], %l2 _____________________________ Hex value in %l2
stb %l2, [%l0+2] ______________________ ____ Hex value in word labeled x
srl %l2, 4, %l2 _______________________ _____ Hex value in %l2
sth %l2, [%l0]

set fmt, %o0
ld [%l0], %o1
call printf ____________________________ Hex value in word labeled x
nop (same as output of this printf)

set i1, %l0
set c, %l1
ldsb [%l1], %l2 ____________________________ Hex value in %l2
sth %l2, [%l0]
sth %l2, [%l0]

set fmt, %o0
ld [%l0], %o1
call printf ____________________________ Hex value in word labeled i1
nop (same as output of this printf)

set i2, %l0
set i3, %l1
ld [%l1], %l2 ____________________________ Hex value in %l2
sth %l2, [%l0]

set fmt, %o0
ld [%l0], %o1
call printf ____________________________ Hex value in word labeled i2
nop (same as output of this printf)

ret
restore
12. Miscellaneous

Complete the truth table for the following logic diagram:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>___</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>___</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>___</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>___</td>
</tr>
</tbody>
</table>

What kind of logic circuit is this? _____ (From table above - answer one of A-F)

Draw the logic circuit for the following boolean expression expressed with C bitwise operators:

\[(a \oplus b) \lor \sim(c \& d)\]

(Use 3 logic gates - Do not use inverters in the logic diagram!)

What value must input D and E be in order to set output Q to the value 1 independent of what value Q may have been previously? Use the letters in the box below to answer this and the next question.

D ____
E ____

What value of E will keep the output Q the same independent of what value D is? ____

What kind of logic circuit is this? _____ (From table above - answer one of A-F)

Given the following program, order the printf() lines so that the values that are printed when run on a Sun SPARC Unix system are displayed from smallest address/value to largest address/value.

```c
#include <stdio.h>
#include <stdlib.h>

void foo( int, int ); /* Function Prototype */

int a = 24;

int main( int argc, char *argv[] )
{
    int b = 99;
    int c;
    foo( c, b );
    /* 1 */ (void) printf( "1: c --> %p\n", &c );
    /* 2 */ (void) printf( "2: foo --> %p\n", foo );
    /* 3 */ (void) printf( "3: b --> %p\n", &b );
    /* 4 */ (void) printf( "4: argc --> %p\n", &argc );
    /* 5 */ (void) printf( "5: malloc --> %p\n", malloc(50) );
}

void foo( int d, int e )
{
    int f = 42;
    static int g;
    /* 6 */ (void) printf( "6: g --> %p\n", &g );
    /* 7 */ (void) printf( "7: e --> %p\n", &e );
    /* 8 */ (void) printf( "8: a --> %p\n", &a );
    /* 9 */ (void) printf( "9: f --> %p\n", &f );
    /* 10 */ (void) printf( "10: d --> %p\n", &d );
}
```

this line prints smallest value

this line prints largest value
**Extra Credit** (10 points)

What is the value of each of the following expressions taken sequentially based on changes that may have been made in previous statements?

```c
#include <stdio.h>

int main()
{
    char a[] = "10019";
    char *ptr = a;
    printf( "%c\n", *ptr++ );            ______
    printf( "%c\n", (*ptr)++ );          ______
    printf( "%c\n", +++ptr );            ______
    printf( "%c\n", +++ptr+ );           ______
    printf( "%c\n", +++ptr++ );          ______
    printf( "%c\n", +++ptr );            ______
    printf( "%c\n", --*++ptr );          ______
    printf( "%d\n", ptr - a );           ______
    printf( "%s\n", a );                _______________
    return 0;
}
```

Given the C array declaration

```c
int a[4][2];
```

Mark with an A the memory location(s) where we would find `a[3][0]`

```
+a:  

```

Each box represents a byte in memory.

What is the color of Rick's eyes?  _____________________________
Hexadecimal - Character

| 00 NUL | 01 SOH | 02 STX | 03 ETX | 04 EOT | 05 ENQ | 06 ACK | 07 BEL |
| 08 BS  | 09 HT  | 0A NL  | 0B VT  | 0C NP  | 0D CR  | 0E SO  | 0F SI  |
| 10 DLE | 11 DC1 | 12 DC2 | 13 DC3 | 14 DC4 | 15 NAK | 16 SYN | 17 ETB |
| 18 CAN | 19 EM  | 1A SUB | 1B ESC | 1C FS  | 1D GS  | 1E RS  | 1F US  |
| 20 SP  | 21 !   | 22 "   | 23 #   | 24 $   | 25 %   | 26 &   | 27 '   |
| 28 (   | 29 )   | 2A *   | 2B +   | 2C ,   | 2D -   | 2E .   | 2F /   |
| 30 0   | 31 1   | 32 2   | 33 3   | 34 4   | 35 5   | 36 6   | 37 7   |
| 38 8   | 39 9   | 3A :   | 3B ;   | 3C <   | 3D =   | 3E >   | 3F ?   |
| 40 @   | 41 A   | 42 B   | 43 C   | 44 D   | 45 E   | 46 F   | 47 G   |
| 48 H   | 49 I   | 4A J   | 4B K   | 4C L   | 4D M   | 4E N   | 4F O   |
| 50 P   | 51 Q   | 52 R   | 53 S   | 54 T   | 55 U   | 56 V   | 57 W   |
| 58 X   | 59 Y   | 5A Z   | 5B [   | 5C \   | 5D ]   | 5E ^   | 5F _   |
| 60 `   | 61 a   | 62 b   | 63 c   | 64 d   | 65 e   | 66 f   | 67 g   |
| 68 h   | 69 i   | 6A j   | 6B k   | 6C l   | 6D m   | 6E n   | 6F o   |
| 70 p   | 71 q   | 72 r   | 73 s   | 74 t   | 75 u   | 76 v   | 77 w   |
| 78 x   | 79 y   | 7A z   | 7B {   | 7C |   | 7D }   | 7E ~   | 7F DEL |

A portion of the Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ postfix increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- postfix decrement</td>
<td></td>
</tr>
<tr>
<td>[] array element</td>
<td></td>
</tr>
<tr>
<td>() function call</td>
<td></td>
</tr>
<tr>
<td>-&gt; struct/union pointer</td>
<td></td>
</tr>
<tr>
<td>. struct/union member</td>
<td></td>
</tr>
<tr>
<td>* indirection</td>
<td>R to L</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td></td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td></td>
</tr>
<tr>
<td>&amp; address-of</td>
<td></td>
</tr>
<tr>
<td>sizeof size of type/object</td>
<td></td>
</tr>
<tr>
<td>(type) type cast</td>
<td></td>
</tr>
<tr>
<td>* multiplication</td>
<td>L to R</td>
</tr>
<tr>
<td>/ division</td>
<td></td>
</tr>
<tr>
<td>% modulus</td>
<td></td>
</tr>
<tr>
<td>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>
Scratch Paper