Signature _____________________  Name _______________________
Login: cs30x_________  Student ID ______________________

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**
**Winter 2014**
**Final Exam**

1. Number Systems / Right-Left Rule _________________ (21 points)
2. Binary Addition/Condition Code Bits/Overflow Detection _________________ (12 points)
3. Branching _________________ (18 points)
4. Bit Operations _________________ (13 points)
5. Recursion/SPARC Assembly _________________ (10 points)
6. Local Variables, The Stack, Return Values _________________ (18 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 _________________ (201 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 0xFAE8 (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 284 to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

sign-magnitude________________________________________________________
1’s complement________________________________________________________
2’s complement________________________________________________________

Convert -659 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 foobar that is a pointer to an array of 17 elements where each element is a pointer to a function that takes a pointer to a struct foo and returns a pointer to a double. (6 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)

\[
\begin{array}{cccc}
11010110 & 00110100 & 00011101
+10111011 & +01001100 & +00101001
\end{array}
\]

\[
\begin{array}{cccc}
N & Z & V & C
\hline
| & | & | & |
\hline
| & | & | & |
\hline
| & | & | & |
\end{array}
\]

\[
\begin{array}{cccc}
N & Z & V & C
\hline
| & | & | & |
\hline
| & | & | & |
\hline
| & | & | & |
\end{array}
\]
3. Branching (18 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. This is not an entire function - just translate the code fragment using the local register mapping indicated.

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

/* Some other code here */

/* Translate just this code below */

if ( a > b )
{
    do
    {
        a = a % b;
    } while ( a >= 55 );

    --b;
}
else
{
    b = b + 44;
}

/* Some other code here */
```

```sparc
  ; a is mapped to %l2
  ; b is mapped to %l5

if a > b
{
    do
    {
        a = a % b;
    } while ( a >= 55 );

    --b;
}
else
{
    b = b + 44;
}
```

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.)

```plaintext
set 0xDEADC0DE, %l0
set 0x86715309, %l1
and %l0, %l1, %l0
```

Value in %l0 is _______________________________________ (2 points)

```plaintext
set 0xDEADC0DE, %l0
sra %l0, 7, %l0
```

Value in %l0 is _______________________________________ (2 points)

```plaintext
set 0xDEADC0DE, %l0
sll %l0, 7, %l0
```

Value in %l0 is _______________________________________ (2 points)

```plaintext
set 0xDEADC0DE, %l0
set 0x????????, %l1
xor %l0, %l1, %l0
! Value in %l0 is now 0x86715309
```

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

```plaintext
set 0xDEADC0DE, %l0
set 0x86715309, %l1
or %l0, %l1, %l0
```

Value in %l0 is _______________________________________ (2 points)

```plaintext
set 0xDEADC0DE, %l0
srl %l0, 6, %l0
```

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

```
.global main
/* main.s */

.section ".rodata"
.align 4
.word 0x30003300
code: .word 0x3E736463, 0x72656D74, 0x6B445061, 0x2F3D4E32, 0x2B206B2B, 0x666B4372
.word 0x602F7545, 0x43537300, 0x00000030

.section ".text"
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
inc %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
```
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
char fubar( int a, char b ) {
    int   local_stack_var1;
    int  *local_stack_var2;
    struct foo {
        int   s1;
        int   s2[3];
        char  s3[3];
        char  s4;
    }     local_stack_var3;
    local_stack_var3.s4 = ++b;  /* 1 */
    local_stack_var1 = ++*local_stack_var2; /* 2 */
    local_stack_var3.s1 = local_stack_var3.s2[0] + a++; /* 3 */
    return ( local_stack_var3.s3[2] ); /* 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.** (18 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 ); // output: x1 = ___
    if ( x1 <= 1 )
    {
        return 1;
    }
    else
    {
        r1 = BBB( x1 - 2 ) + x1;
        printf( "r1 = %d\n", r1 ); // output: r1 = ___
        return r1;
    }
}

int BBB( int x2 )
{
    int r2;
    printf( "x2 = %d\n", x2 ); // output: x2 = ___
    if ( x2 <= 2 )
    {
        return 2;
    }
    else
    {
        r2 = AAA( x2 - 1 ) + x2;
        printf( "r2 = %d\n", r2 ); // output: r2 = ___
        return r2;
    }
}

int main( int argc, char *argv[] )
{
    // output value returned by
    // initial call
    printf( "%d\n", AAA( 5 ) );
    return 0;
}
```

Put output here
8. Floating Point

Convert -139.25_{10} (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

binary fixed-point __________________________________   (2 points)
IEEE floating-point __________________________________   (4 points)

Convert 0x43356000 (single-precision IEEE floating-point representation) to fixed-point decimal.

fixed-point decimal __________________________________   (6 points)

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).

```
addcc  %l3, -9, %i2
sth    %o2, [%i3 + %l1]
```

addcc  %l3, -9, %i2 ___________________________________________   (5 points)
sth    %o2, [%i3 + %l1] _________________________________________   (5 points)

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

```
0x2E800008
0xAC1AC019
```

0x2E800008 _________________________________________   (5 points)
0xAC1AC019 _________________________________________   (5 points)

If an odd-ball computer had 121MB 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)

static int a; ______________ (a)
int b = 42; ______________ (b)
int c; ______________ (c)
static int d = 11; ______________ (d)
static int foo( int e ) { ____________ (e) ____________ (foo)
    float f = 42.24; ______________ (f)
    char *g;
    g = (char *) malloc( b ); ______________ (where g is pointing)
    static int (*h)(int) = foo; ____________ (h) ____________ (where h is pointing)
    static double i;
    ____________ (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 ________ 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 5 times, indicate how many times will f be initialized? ________
If the function foo() above is called 5 times, indicate how many times will h 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.)

.global main

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

.c: .byte 0x99
.s: .half 0xFACE
.i1: .word 0x9ABCD123
.i2: .word 0x9ABCD123
.i3: .word 0x9ABCD123
.x: .word 0x55550000

.align 2

.s: .half 0xFACE

.align 4

.i1: .word 0x9ABCD123
.i2: .word 0x9ABCD123
.i3: .word 0x9ABCD123
.x: .word 0x55550000

.section ".text"

main:
save   %sp, -96, %sp

set   x, %l0
set   s, %l1
lduh  [%l1], %l2   Hex value in %l2
stb   %l2, [%l0+3]   Hex value in word labeled x
srl   %l2, 4, %l2   Hex value in %l2
stb   %l2, [%l0+1]

set   fmt, %o0
ld    [%l0], %o1
call  printf
nop   (same as output of this printf)

set   i1, %l0
set   c, %l1
ldsb  [%l1], %l2   Hex value in %l2
stb   %l2, [%l0]
stb   %l2, [%l0+3]

set   fmt, %o0
ld    [%l0], %o1
call  printf
nop   (same as output of this printf)

set   i2, %l0
set   i3, %l1
ld    [%l1], %l2   Hex value in %l2
stb   %l2, [%l0+1]   Hex value in word labeled i2
sra   %l2, 12, %l2   Hex value in %l2
stb   %l2, [%l0+2]

set   fmt, %o0
ld    [%l0], %o1
call  printf
nop   (same as output of this printf)

ret
restore
12. Miscellaneous

What value must input D and E be in order to set output Q to the value 0 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 ___

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>Either 0 or 1</td>
</tr>
</tbody>
</table>

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 below - answer one of A-F)

Complete the truth table for the following logic diagram:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</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:

\[(a \& b) \oplus \neg(c | d)\]

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

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 */
static int a;

int main( int argc, char *argv[] ) {
    int b = 7;
    static int c = 11;

    foo( argc, c );

    /*  1 */ (void) printf( "1: argc --> %p\n", &argc );
    /*  2 */ (void) printf( "2: malloc --> %p\n", malloc(50) );
    /*  3 */ (void) printf( "3: foo --> %p\n", foo );
    /*  4 */ (void) printf( "4: b --> %p\n", &b );
    /*  5 */ (void) printf( "5: c --> %p\n", &c );
}

void foo( int d, int e ) {
    int f = 5;
    int g;

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

this line prints

---

this line prints

---

---

---

---

---

---

---

---

---

---

---

---

---

---
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[] = "CSE30";
    char *ptr = a;

    printf( "%c
", *ptr++ );
    printf( "%c
", (*ptr)++ );
    printf( "%c
", +++ptr );
    printf( "%c
", +++ptr++ );
    printf( "%c
", +++ptr );
    printf( "%c
", --*++ptr );
    printf( "%d
", ptr - a );
    printf( "%s
", 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[2][1]
```

Each box represents a byte in memory.

What is Rick's favorite Disney movie? _____________________________
### Hexadecimal - Character

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

### A portion of the Operator Precedence Table

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