CSE 30
Spring 2006
Final Exam

1. Number Systems ___________________ (15 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 and Return Values ___________________ (27 points)
7. Bit Slinging ___________________ (12 points)
8. Floating Point ___________________ (12 points)
9. Machine Instructions ___________________ (20 points)
10. Linkage, Scope, Lifetime, Data ___________________ (34 points)
11. Load/Store/Memory ___________________ (9 points)
12. Miscellaneous ___________________ (33 points)

SubTotal ___________________ (215 points)
Extra Credit ___________________ (10 points)
Total ___________________
1. Number Systems
Convert \texttt{0xFA5A} (2’s complement, 16-bit word) to the following. (6 points)

- **binary** ____________________________ (straight bit pattern translation)
- **octal** ____________________________ (straight bit pattern translation)
- **decimal** ____________________________ (pos/neg decimal value from 2’s complement encoding)

Convert \texttt{377}_{10} to the following (assume 16-bit word). \textbf{Express answers in hexadecimal.} (3 points)

- **sign-magnitude** ____________________________
- **1’s complement** ____________________________
- **2’s complement** ____________________________

Convert \texttt{-652}_{10} to the following (assume 16-bit word). \textbf{Express answers in hexadecimal.} (6 points)

- **sign-magnitude** ____________________________
- **1’s complement** ____________________________
- **2’s complement** ____________________________

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)

\[
\begin{array}{c}
01010101 \\
+01101011 \\
\hline
\end{array} \\
\begin{array}{c}
10110110 \\
+01001010 \\
\hline
\end{array} \\
\begin{array}{c}
00011011 \\
+01011001 \\
\hline
\end{array}
\]

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

Write the SPARC assembly statements to perform the following C statements. **Do not optimize.** (18 points)

```c
int x;
int y;

if ( x < 15 && y >= 35 )
{
    x = x + y;
} else {
    x = fubar( y / 7 );
}
```
4. Bit Operations

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

set $0x2006ACED, %l0
set $0x98675309, %l1
or %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set $0x2006ACED, %l0
sll %l0, 13, %l0

Value in %l0 is _______________________________________ (2 points)

set $0x2006ACED, %l0
srl %l0, 5, %l0

Value in %l0 is _______________________________________ (2 points)

set $0x2006ACED, %l0
set $?????????, %l1
xor %l0, %l1, %l0 ! Value in %l0 is now OxDEADBEEF

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

set $0x2006ACED, %l0
set $0x98675309, %l1
and %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set $0x2006ACED, %l0
sra %l0, 13, %l0

Value in %l0 is _______________________________________ (2 points)
5. Recursion/SPARC Assembly
Given `main.s` and `recurse.s`, what gets printed when executed? (10 points)

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

.section ".rodata"
.align 4
.code: .word 0x30003300
.word 0x33736447, 0x72657074, 0x6654357300, 0x0000030

.main:
    save %sp, -92 & -8, %sp
    set code, %o0
    mov 27, %o1
    call fubar
    nop
    ret
    restore

.global fubar /* fubar.s */

.fmt: .asciz "%c"

.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
```

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

```c
long fubar( char *x, int y ) {
    char  *local_stack_var1;
    struct foo {
        char  s1[5];
        char  s2;
        char  *s3;
        int   s4;
    }       local_stack_var2;
    char  local_stack_var3;

    x = local_stack_var2.s3; /* 1 */
    local_stack_var3 = local_stack_var2.s1[3] + 3; /* 2 */
    local_stack_var1 = &local_stack_var2.s2; /* 3 */

    return ( y + *++local_stack_var1 ); /* 4 */
}
```

Now 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. (27 points)
7. Bit Slinging

Write a C function named evenOrOddNumOfBits() that takes a word (unsigned int) and returns 0 if there are an even number of 1's in the word and returns 1 if there are an odd number of 1's in the word. (12 pts)

For example,  
\[
\text{evenOrOddNumOfBits}( 0x7FFFFFFF ) \quad \text{will return the value 1} \\
\text{evenOrOddNumOfBits}( 0x00201000 ) \quad \text{will return the value 0}
\]

C

\[
\text{unsigned int} \\
\text{evenOrOddNumOfBits( unsigned int word )}
\]
8. Floating Point

Convert \(-124.625\) (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Result</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 0x43446000 (single-precision IEEE floating-point representation) to fixed-point decimal.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Result</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 number of instructions away for the target (vs. a Label).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>std %o2, [%i4 + %i3]</td>
<td>__________________________</td>
</tr>
<tr>
<td>xor %i5, -15, %i3</td>
<td>__________________________</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x92848006</td>
<td>__________________________</td>
</tr>
<tr>
<td>0xE857BFF6</td>
<td>__________________________</td>
</tr>
</tbody>
</table>
For the following program fragment, specify what C runtime area/segment will be used for each variable definition or statement: (34 points — 1 point each)

```c
int a = 411;
static int b;
int c;
static int d = 420;

int foo( int e ) {  
    static double f = -3.33;
    static int *g;
    int h = 611;

    int (*i)(int) = foo;
    g = (int *) malloc( d );
    ...
}
```

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.
B) Global just to this source file.
C) Local to function foo().

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If function foo() is called 7 times, how many times does the variable f get initialized to -3.33? _______

If function foo() is called 7 times, how many times does the variable h get initialized to 611? _______

---

10. Linkage, Scope, Lifetime, Data

For the following program fragment, specify what C runtime area/segment will be used for each variable definition or statement: (34 points — 1 point each)
11. Load/Store/Memory
What gets printed in the following program? (9 points)

```
.global main

.section ".data"
fmt:    .asciz "0x%8x\n"         ! prints value as hex  0XXXXXXXX

c:      .byte   0x88

.salign 2
s:      .half   0xABCD

.salign 4
i1:     .word   0x98675309
i2:     .word   0x98675309
i3:     .word   0x98675309
x:      .word   0x00004321

.section ".text"
main:

save    %sp, -96, %sp

set     i1, %l0
set     s, %l1

ldsh    [%l1], %l1
st      %l1, [%l0]
stb     %l1, [%l0+1]

set     fmt, %o0
ld      [%l0], %o1
call    printf   __________________________
nop

set     i2, %l0
set     c, %l1

ldsb    [%l1], %l1
stb     %l1, [%l0+2]

set     fmt, %o0
ld      [%l0], %o1
call    printf   __________________________
nop

set     x, %l0
set     i3, %l1

ldsb    [%l1], %l2
stb     %l2, [%l0+2]

set     fmt, %o0
ld      [%l0], %o1
call    printf   __________________________
nop

ret
```
12. Miscellaneous

Put the following in the correct order/sequence using the numbers to the left of each word. (9 pts)

1. C preprocessor 4. assembler 7. C source code
2. program execution 5. compiler 8. executable (.exe/a.out)
3. Core Dump (Segmentation Fault) 6. linker/linkage editor 9. loader

---\-> ---\-> ---\-> ---\-> ---\-> ---\-> ---\-> ---\-> ---\->

Draw the logic circuit to perform the following Boolean logic. Label each gate. Do not optimize. (3 pts)

!(a & b) ^ (!a | b)

Is this a sequential or combinational logic circuit? (Circle correct answer in the question to the left.) (1 pt)

When the following program is run on a Sun SPARC Unix system and sorted by the address printed with the \%p format specifier, specify the order of the lines printed from smallest value to largest value. (2 points each)

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

int a = 43;

int main( int argc, char *argv[]) {
    int b = 7;
    int c = 11;
    foo( argc, b);
    /* 1 */ (void) printf( "1: b --> %p\n", &b );
    /* 2 */ (void) printf( "2: malloc --> %p\n", malloc(50) );
    /* 3 */ (void) printf( "3: c --> %p\n", &c );
    /* 4 */ (void) printf( "4: argc --> %p\n", &argc );
    /* 5 */ (void) printf( "5: foo --> %p\n", foo );
}

void foo( int d, int e ) {
    int f;
    static int g;
    /* 6 */ (void) printf( "6: f --> %p\n", &f );
    /* 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: g --> %p\n", &g);
}

Extra Credit
Optimize the following assembly code fragment. (up to 3 pts)

```
mov   %l2, %o0
mov   64, %o1
call  .mul
nop
mov   %o0, %l2
```

What is the value of each of the following expressions? (1 pt each)

```
char *a = "End this, please!"; /* char a[] = "End this, please!"; */
toupper( a[strlen( a ) - 2] + 1 ) _______
0["I loved CSE 30!"] _______
*a + 1 _______
*("This Blows Me Away!" + 14) _______
toupper( "This Blows Me Away!"[8] ) _______
*"Damn this is weird!" - 1 _______
```

Tell me something you learned in this class that is extremely valuable to you and that you think you will be able to use for the rest of your programming/computer science career. (1 point if serious; you can add non-serious comments also)
### 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>
Scratch Paper