CSE 30
Winter 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, 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 0xFA4A (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 388\textsubscript{10} to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

sign-magnitude __________________________
1’s complement __________________________
2’s complement __________________________

Convert -713\textsubscript{10} to the following (assume 16-bit word). **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)

\[
\begin{array}{c}
\begin{array}{c}
11010101 \\
+10101011 \\
\hline \\
N \quad Z \quad V \quad C
\end{array} & \begin{array}{c}
11010110 \\
+01010100 \\
\hline \\
N \quad Z \quad V \quad C
\end{array} & \begin{array}{c}
00111011 \\
+01111001 \\
\hline \\
N \quad Z \quad V \quad C
\end{array}
\end{array}
\]
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 > 25 )
{
    y = foo( x );
    x = x + y;
} else {
    x = y - x;
    x = x / 7;
}
```
4. Bit Operations

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

set 0xCAFEBABE, %l0
set 0x86753099, %l1
or %l0, %l1, %l0

Value in %l0 is ____________________________ (2 points)

set 0xCAFEBABE, %l0
sll %l0, 14, %l0

Value in %l0 is ____________________________ (2 points)

set 0xCAFEBABE, %l0
srl %l0, 7, %l0

Value in %l0 is ____________________________ (2 points)

set 0xCAFEBABE, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0 ! Value in %l0 is now 0x420FADED

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

set 0xCAFEBABE, %l0
set 0x86753099, %l1
and %l0, %l1, %l0

Value in %l0 is ____________________________ (2 points)

set 0xCAFEBABE, %l0
sra %l0, 9, %l0

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

```
.globl main  /* main.s */
.sect .section ".rodata"
.align 4
.code: .word 0x43216453, 0x21654567, 0x6B336E61, 0x306F4E20, 0x4C205220, 0x66756572
        .word 0x6C767565, 0x6953734C, 0x00000030
.sect ".text"
main:
    save %sp, -92 & -8, %sp
    set code, %o0
    mov 1, %o1
    call fubar
    nop
    ret
restore

.globl fubar  /* fubar.s */
.sect ".rodata"
fmt: .asciz "%c"
.sect ".text"
fubar:
    save %sp, -(92 + 1) & -8, %sp
    inc %i1
    cmp %i0, %g0
    be end
    nop
    ldub [%i0 + %i1], %l0
    cmp %l0, %g0
    be end
    nop
    stb %l0, [%fp - 1]
    add %i1, 2, %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
short fubar( long x, char *y ) {
    short local_stack_var1;
    struct foo {
        short s1[3];
        char *s2;
        short s3;
        int s4;
    } local_stack_var2;
    short *local_stack_var3;
    local_stack_var1 = local_stack_var2.s1[1] + 13; /* 1 */
    local_stack_var2.s2 = y; /* 2 */
    local_stack_var3 = &local_stack_var2.s4; /* 3 */
    return ( x + *local_stack_var3++ ); /* 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 flipBit() that takes a word (unsigned int) and a bit position and returns the word with the nth bit flipped/toggled. The bits in the word are numbered 0-31 from right to left (bit 0 is the least significant bit; bit 31 is the most significant bit). (12 pts)

For example, flipBit( 0xFFFFFFFF, 0 ) will return the value 0xFFFFFFFFE
flipBit( 0x00000000, 31 ) will return the value 0x80000000

C
unsigned int
flipBit( unsigned int word, unsigned int n )
8. Floating Point

Convert 129.375\textsubscript{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 0xC2E4E000 (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 number of instructions away for the target (vs. a Label).

save %sp, -96, %sp ________________________________ (5 points)

or %g5, -12, %o3 ________________________________ (5 points)

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

0xF4330013 ________________________________ (5 points)

0x2ABFFFFC ________________________________ (5 points)
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)

```c
static int a = 411;
static int b;
int c;
int d = 420;
static int foo( int e ) {
    double f = -3.33;
    static int *g;
    static 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().

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 function foo() is called 7 times, how many times does the variable h get initialized to 611? _______
If function foo() is called 7 times, how many times does the variable f get initialized to -3.33? _______
11. Load/Store/Memory
What gets printed in the following program? (9 points)

.global main

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

.c: .byte 0x77

.s: .half 0x9145

.i1: .word 0xCAFEBAEB
.i2: .word 0xCAFEBAEB
.i3: .word 0xCAFEBAEB
.x: .word 0x5419

.section ".text"
main:
save %sp, -96, %sp

set i1, %10
set s, %11

ldsh [%ll], %11
st %ll, [%10]
stb %ll, [%10+1]

set fmt, %o0
ld [%10], %o1
call printf

nop

set i2, %10
set c, %11

ldsb [%ll], %11
sth %ll, [%10+2]
stb %ll, [%10+1]

set fmt, %o0
ld [%10], %o1
call printf

nop

set x, %10
set i3, %11

ldsb [%ll+1], %12
sth %12, [%10]
stb %12, [%10+3]

set fmt, %o0
ld [%10], %o1
call printf

nop

ret
restore
12. Miscellaneous

Put the following in the correct order/sequence using the numbers to the left of each word. (8 pts)

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

_____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____

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

\[(a \land !b) & !(a \lor b)\]

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

What is Rick's favorite TV show (as determined by the class at the end of the last day of classes)? (1 pt)

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 value to largest value. (2 points each)

```c
void foo( int, int ); /* Function Prototype */
int a;
int main( int argc, char *argv[] ) {
    static int b = 7;
    int c = 11;
    printf( "malloc --> %p\n", malloc(50) );
    /* 1 */ (void) printf( "c --> %p\n", &c );
    /* 2 */ (void) printf( "argc --> %p\n", &argc );
    /* 3 */ (void) printf( "foo --> %p\n", foo );
    /* 4 */ (void) printf( "b --> %p\n", &b );
    /* 5 */ (void) printf( "argc --> %p\n", &argc );
    printf( "malloc --> %p\n", malloc(50) );
    /* 6 */ (void) printf( "a --> %p\n", &a );
    /* 7 */ (void) printf( "e --> %p\n", &e );
    /* 8 */ (void) printf( "d --> %p\n", &d );
    /* 9 */ (void) printf( "g --> %p\n", &g );
    /* 10 */ (void) printf( "f --> %p\n", &f );
}
void foo( int d, int e ) {
    int f = 911;
    int g;
    /* 6 */ (void) printf( "a --> %p\n", &a );
    /* 7 */ (void) printf( "e --> %p\n", &e );
    /* 8 */ (void) printf( "d --> %p\n", &d );
    /* 9 */ (void) printf( "g --> %p\n", &g );
    /* 10 */ (void) printf( "f --> %p\n", &f );
```
Extra Credit
What does the following SPARC assembly language program output?

```assembly
.global main

.code: .asciz "102"
fmt: .asciz "%s = %d\n"

.main:
save %sp, -96, %sp
set code, %l0
ldub [%l0], %l1
clr %l2
cmp %l1, %g0
be L1
nop

L2:
mov %l2, %o0
mov 8, %o1
call .mul
nop

ldub [%l0], %l1
sub %l1, 0x30, %l1
add %o0, %l1, %l2
inc %l0
ldub [%l0], %l1
cmp %l1, %g0
bne L2
nop

L1:
set fmt, %o0
set code, %o1
mov %l2, %o2
call printf
nop
ret
restore
```

Output ________________________ (2 points)

Now optimize the code to get the same result with the fewest cycles. Some optimizations are better than others. You may not be able to eliminate all nops. Go for the fewest machine cycles assuming memory accesses are several more cycles than other non-memory access instructions. You cannot change the overall algorithm. (8 points)
<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>7D</td>
<td>7E ~</td>
<td>7F DEL</td>
</tr>
</tbody>
</table>
Scratch Paper