Name _________________________
Signature_______________________
cs30x_______

CSE 30
Spring 2002
Final Exam

1. Number Systems ___________________  (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching ___________________  (20 points)
4. Bit Operations ___________________  (10 points)
5. Recursion/SPARC Assembly ___________________  (10 points)
6. Local Variables, The Stack, and Return Values ___________________  (20 points)
7. SPARC Subroutines and Calling Convention ___________________  (14 points)
8. Floating Point ___________________  (12 points)
9. Linkage, Scope, Lifetime, Data ___________________  (31 points)
10. Machine Instructions ___________________  (20 points)
11. Bits and Pieces ___________________  (12 points)
12. Miscellaneous ___________________  (27 points)

SubTotal ___________________  (203 points)
Extra Credit ___________________  (10 points)
Total ___________________
1. Number Systems

Convert \textit{F9DA}	extsubscript{16} (2’s complement, 16-bit word) to the following. (6 points)

- binary_______________________________________
- octal  _______________________________________
- decimal ______________________________________

Convert \textit{-698}	extsubscript{10} to the following (assume 16-bit word). \textbf{Express answers in hexadecimal}. (6 points)

- sign-magnitude____________________________________
- 1’s complement____________________________________
- 2’s complement____________________________________

Convert \textit{+537}	extsubscript{10} to the following (assume 16-bit word). \textbf{Express answers in hexadecimal}. (3 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)

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>10111001</td>
<td>00101001</td>
<td>10010101</td>
<td>00101011</td>
<td>11010110</td>
<td>11010111</td>
<td>10111011</td>
<td></td>
</tr>
</tbody>
</table>

3. Branching

Write the C statements to perform the following SPARC assembly statements. Do not optimize. (20 points)

```c
x = 40411;
while ( x > 9 ) {
    if ( (x % 16) < 4 )
        x = x - 19;
    else
        x = x - 45;
}
```
4. Bit Operations

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

```plaintext
set 0xE9F436CB, %l0
set 0xCD59CD78, %l1
xor %l0, %l1, %l0

Value in %l0 is ________________________________  (2 points)

set 0xE9F436CB, %l0
sll %l0, 9, %l0

Value in %l0 is ________________________________  (2 points)

set 0xE9F436CB, %l0
sra %l0, 14, %l0

Value in %l0 is ________________________________  (2 points)

set 0xE9F436CB, %l0
set 0x91817161, %l1
bclr %l1, %l0

Value in %l0 is ________________________________  (2 points)

set 0xE9F436CB, %l0
set 0x8276BD15, %l1
or %l0, %l1, %l0

Value in %l0 is ________________________________  (2 points)
```
5. Recursion/SPARC Assembly

Given `main.s` and `fubar.s`, what gets printed when executed? (10 points)

```assembly
/* main.s */
.globl main

.sect .rodata
.align 2

.code:
.half 0x4361, 0x2153, 0x7664, 0x4561, 0x6533, 0x4A6B
.half 0x3020, 0x6120, 0x3D4E, 0x5221, 0x206F, 0x2066
.half 0x6323, 0x726B, 0x4375, 0x7300, 0x5300, 0x3000

.sect .text
main:
    save    %sp, -92 & -8, %sp
    set     code, %o0
    mov     0, %o1
    call    fubar
    nop
    ret
    restore

/* fubar.s */
.globl fubar

.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
```

```
6. Local Variables, The Stack, and Return Values
Here is a C function that allocates a couple 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 x, char y ) {
    short  local_stack_var1;
    struct foo {
        int     s1;
        char    s2;
        unsigned short  s3[5];
        long    s4;
    }      local_stack_var2;
    int    *local_stack_var3;
    local_stack_var2.s4 = *local_stack_var3++;    /* 1 */
    local_stack_var2.s2 = y;                  /* 2 */
    local_stack_var1 = local_stack_var2.s3[3];  /* 3 */
    return ( x + local_stack_var2.s1 );          /* 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. (20 points)
7. SPARC Subroutines and Calling Convention (2 point each)

In most architectures, some of the Stack Frame is built by the calling function and some of the Stack Frame is built by the called function. The same is generally true of the SPARC architecture. Fill in the blanks for the general model of Stack Frame creation/function calling convention.

The ______________ function places the return value in the architecture-specific return value area.

The ______________ function deallocates the space on the Stack used for the arguments.

The ______________ function allocates space on the Stack for local variables and performs any initialization.

The ______________ function calls the instruction to set the program counter with the saved return address.

The ______________ function allocates space on the Stack for the arguments being passed in the function call.

The ______________ function calls the instruction to save the current value of the program counter to be used as the return address.

The ______________ function allocates space for and possibly saves values of registers that it may use.

8. Floating Point

Convert $123.375_{10}$ (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 0xC2BD4000 (single-precision IEEE floating-point representation) to fixed-point decimal.

fixed-point decimal __________________________________   (6 points)
9. Linkage, Scope, Lifetime, Data

For the following program fragment, specify what C runtime area/segment will be used for each variable definition or statement: (31 points — 1 point each)

```c
int a;
static int b = 4;
static int c;
int d = 74;
static int foo(int e) {
    double f = 4.20;
    static int *g;
    int *h;
    static int i = 420;
    g = (int *) malloc(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.
B) Global just to this source file.
C) Local to function foo().

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.

```
a _____
b _____
c _____
d _____
e _____
f _____
g _____
h _____
i _____
foo _____
```
10. 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).

\[
\text{subcc} \quad \%i3, -10, \%o3 \quad \text{__________________________} \quad (5 \text{ points})
\]

\[
\text{ldsh} \quad [\%i4], \%l2 \quad \text{__________________________} \quad (5 \text{ points})
\]

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

\[
0x24BFFFF9 \quad \text{__________________________} \quad (5 \text{ points})
\]

\[
0xB68D000A \quad \text{__________________________} \quad (5 \text{ points})
\]

11. Bits and Pieces

Fill in the blanks to complete this function which takes a SPARC Format 3 instruction (bit pattern) as the parameter and returns the bit pattern of the second source operand. (6 points)

```c
/*
 * If the 2nd src operand is an immediate constant, return the
 * bit pattern of the immediate constant, otherwise return the bit pattern
 * of the source register 2 encoding.
 */
unsigned int getOperand2Bits1( unsigned int instr ) {
    unsigned int iBitMask = _________________________;
    unsigned int immedConstMask = _________________________;
    unsigned int regSrc2Mask = _________________________;
    if ( instr & iBitMask )
        return instr & immedConstMask;
    else
        return instr & regSrc2Mask;
}
```

Now do the same thing using only bit shift operations. (6 points)

```c
unsigned int getOperand2Bits2( unsigned int instr ) {
    if ( ____________________________ )
        return ____________________________;
    else
        return ____________________________;
}
```
12. Miscellaneous

What is the default buffering for (1 pt each)

<table>
<thead>
<tr>
<th>Buffering</th>
<th>File I/O</th>
<th>____________</th>
<th>stderr</th>
<th>____________</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>stdin</td>
<td>____________</td>
<td>stdout</td>
<td>____________</td>
</tr>
</tbody>
</table>

List one way to generally reduce Seek Time when performing disk I/O. (1 pt)

What is RISC an acronym for? (1 pt)

What is CISC an acronym for? (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 );

int a;

int main( int argc, char *argv[] ) {
    static int b = 404;
    int c = 911;
    
    foo( c, b );
    
    /* 1 */ (void) printf( "argc --> %p\n", &argc );
    /* 2 */ (void) printf( "main --> %p\n", main );
    /* 3 */ (void) printf( "malloc --> %p\n", malloc(50) );
    /* 4 */ (void) printf( "b --> %p\n", &b );
    /* 5 */ (void) printf( "a --> %p\n", &a );
    /* 6 */ (void) printf( "c --> %p\n", &c );
}

void foo( int d, int e ) {
    int f;
    int g = 101;
    
    /* 7 */ (void) printf( "d --> %p\n", &d );
    /* 8 */ (void) printf( "f --> %p\n", &f );
    /* 9 */ (void) printf( "e --> %p\n", &e );
    /* 10 */ (void) printf( "g --> %p\n", &g );
}
```
Extra Credit1
Optimize the SPARC assembly code of the module fubar.s from problem #5. Recursion/SPARC Assembly (pg 5) to eliminate any nops. You cannot change the overall flow/algorithm. Assume everything above the fubar: label is the same as what is on page 5. (2 points)

fubar:

Extra Credit2
Optimize the SPARC assembly code you wrote for problem #3. Branching (pg 3). Some optimizations are better than others and are therefore worth more points. You cannot change the overall flow/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 0</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 0</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
Scratch Paper