Name _________________________
Signature_______________________

CSE 30
Spring 2001
Final Exam

1. Number Systems ___________________  (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching ___________________  (21 points)
4. Bit Operations ___________________  (10 points)
5. Recursion/SPARC Assembly ___________________  (20 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. Linking/Compiling ___________________  (12 points)
12. I/O, ALU, Control Unit, Registers ___________________  (16 points)

SubTotal ___________________  (203 points)

Extra Credit / C Tricks ___________________  (10 points)

Total ___________________
1. Number Systems

Convert $EDA_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

binary_____________________________________

octal ________________________________

decimal ________________________________

Convert $+358_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

sign-magnitude__________________________________________

1’s complement________________________________________

2’s complement________________________________________

Convert $-297_{10}$ 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)

```
11000101
+01111011
---------
```

```
00111001
+01010111
---------
```

```
10111001
+01010110
---------
```

<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

Fill in the SPARC assembly instructions to perform the following statements. Do not optimize. (17 points)

```
C

int x = 14;                   ! x mapped to %l0

do {
    statement1;
    if ( x <= 8 ) {
        --x;
        statement2;
    } else
        statement3;
    x += -4;
} while ( x < -9 );
```

For the following instruction sequence, mark with an X the conditional branch instructions which would transfer control to loop if used in place of ba. (4 points)

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>bcs</th>
<th>bpos</th>
<th>bvs</th>
<th>bne</th>
<th>ble</th>
<th>bge</th>
<th>bneg</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov 8, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addcc %10, -5, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Bit Operations

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

set 0xAB437EF9, %l0
set 0x56DCDC34, %l1
xor %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xAB437EF9, %l0
sll %l0, 11, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xAB437EF9, %l0
sra %l0, 9, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xAB437EF9, %l0
set 0xF1213121, %l1
btog %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xAB437EF9, %l0
set 0x928296B2, %l1
bset %l1, %l0

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

Given main.s and fubar.s, what gets printed when executed? (20 points)

```assembly
.global main
.sect .".rodata"
code: .byte 0x43, 0x6D, 0x53, 0x73, 0x45, 0x41, 0x20, 0x2D, 0x52, 0x43
 .byte 0x75, 0x52, 0x6C, 0x41, 0x65, 0x50, 0x73, 0x53, 0x00, 0x00
.sect .".text"
main:
save %sp, -96, %sp
set code, %o0
mov 16, %o1
call fubar
nop
ret
restore %g0, %g0, %o0

.sect .".rodata"
fmt: .asciz "%c"
.sect .".text"
fubar:
save %sp, -(92 + 1) & -8, %sp
andcc %i1, 1, %g0
bne else
nop
cmp %i1, %g0
bl end
nop
ldub [%i0+%i1], %l0
stb %l0, [%fp-1]
mov %i0, %o0
sub %i1, 2, %o1
call fubar
nop
set fmt, %o0
ldub [%fp-1], %o1
call printf
nop
ba end
nop
else:
ldub [%i0+%i1], %l0
cmp %l0, %g0
be end
nop
stb %l0, [%fp-1]
mov %10, %o0
add %i1, 2, %o1
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 variable, performs some assignments and returns a value. Do not worry about any local variables not being initialized before being used. Just do a direct translation.

```c
int fubar( int a, int b ) {
    unsigned short local_stack_var1;
    struct foo {
        int s1;
        short s2[5];
        int s3;
        char s4;
    } local_stack_var2;
    int *local_stack_var3;
    local_stack_var1 = *++local_stack_var3;
    local_stack_var2.s4 = 'Q'; /* Use the ASCII value, not 'Q' */
    local_stack_var2.s2[2] = local_stack_var1;
    return ( local_stack_var2.s3 + b );
}
```

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)
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 call convention.

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

The __________ instruction saves the return address.

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

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

The __________ instruction uses the saved return address to set the program counter.

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

The __________ function deallocates the space of the Stack used for the arguments.

**8. Floating Point**

Convert 151.875\(_{10}\) (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 0xC29A4000 (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
static int a;

int b = 4;

static int c = 20;

int d;

static int foo( int e ) {

    static float f = 4.20;

    int g = 420;

    static int h;

    int *i;

    i = (int *) malloc( 7 );

    ...
}
```

Fill in the letter corresponding to the correct scoping/visibility for each of the variables:   lifetime 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>Variable</th>
<th>Scope/Visibility</th>
<th>Lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>b</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>c</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>d</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>e</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>f</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>g</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>h</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>i</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>foo</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>
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).

srl %o3, 14, %l4 ___________________________________  (5 points)
sth %i2, [%fp + %l2] ___________________________________  (5 points)

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

0x3EBFFF52 ___________________________________  (5 points)
0xD41CA010 ___________________________________  (5 points)

11. Linking/Compiling

List two advantages of Static Linking over Dynamic Linking. (4 points)

1) 

2) 

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

1. loader 5. program execution
2. executable (.exe/a.out) 6. assembler
3. compiler 7. preprocessor
4. source code 8. linker

______ → ______ → ______ → ______ → ______ → ______ → ______ → ______
12. I/O, ALU, Control Unit, Registers

Use the terms listed below to fill in the blanks. Terms may be used more than once. (2 points each)

________________________ is noted for using special commands to perform I/O via specially named I/O ports.

When a process performs I/O to an external device, the system will perform a ______________________ on that process while the I/O operation is being performed so other processes can have time on the CPU.

The type of I/O in the preceding question is ______________________ since the I/O device will need to notify the CPU when the I/O request has completed.

After the CPU Registers, the next fastest memory is ______________________.

________________________ usually involves polling a device to determine if an I/O has completed.

One of the fastest ways to perform block I/O is via burst-mode ________________________________.

Recently translated virtual to physical addresses are stored in the _______________________________.

A _____________________________________ ensures main memory is the same as the cache copy.

<table>
<thead>
<tr>
<th>Tri-State device(s)</th>
<th>condition code bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write-Back cache</td>
<td>L1 cache</td>
</tr>
<tr>
<td>Arithmetic Logic Unit (ALU)</td>
<td>L2 cache</td>
</tr>
<tr>
<td>CPU Register(s)</td>
<td>Translation Lookaside Buffer (TLB)</td>
</tr>
<tr>
<td>Program Counter (PC)</td>
<td>Memory Management Unit (MMU)</td>
</tr>
<tr>
<td>Instruction Register (IR)</td>
<td>Input enable</td>
</tr>
<tr>
<td>Bus(es)</td>
<td>Output enable</td>
</tr>
<tr>
<td>Control Signal(s)/Line(s)</td>
<td>Combinational logic circuit/gate(s)</td>
</tr>
<tr>
<td>Process Status Register (PSR)</td>
<td>Sequential logic circuit/gate(s)</td>
</tr>
<tr>
<td>Pipeline/pipelining</td>
<td>Synchronous/synchronizing</td>
</tr>
<tr>
<td>Decode/decoding</td>
<td>Asynchronous/asynchronizing</td>
</tr>
<tr>
<td>Encode/encoding</td>
<td>Write-Through cache</td>
</tr>
<tr>
<td>Context switch</td>
<td>Block Buffering</td>
</tr>
<tr>
<td>Memory-Mapped I/O</td>
<td>I/O Mapped</td>
</tr>
<tr>
<td>I/O Processor</td>
<td>DMA</td>
</tr>
<tr>
<td>Programmed I/O</td>
<td>Interrupt-Driven I/O</td>
</tr>
</tbody>
</table>
Extra Credit

Write a short C program to determine if the underlying architecture is Big-Endian or Little-Endian. Do not assume any underlying data size other than a char is 1 byte (hint: use sizeof()). (5 points)

Write a short C program to determine if the Stack of the underlying architecture grows toward high memory or low memory. You cannot assume local variables or parameters are allocated in any particular order. You cannot assume local variables and parameters within a Stack Frame have a known relative relationship with each other. (5 points)
<table>
<thead>
<tr>
<th>Hexadecimal - Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 NUL</td>
</tr>
<tr>
<td>08 BS</td>
</tr>
<tr>
<td>10 DLE</td>
</tr>
<tr>
<td>18 CAN</td>
</tr>
<tr>
<td>20 SP</td>
</tr>
<tr>
<td>28 (</td>
</tr>
<tr>
<td>30 0</td>
</tr>
<tr>
<td>38 8</td>
</tr>
<tr>
<td>40 \</td>
</tr>
<tr>
<td>48 H</td>
</tr>
<tr>
<td>50 P</td>
</tr>
<tr>
<td>58 X</td>
</tr>
<tr>
<td>60 `</td>
</tr>
<tr>
<td>68 h</td>
</tr>
<tr>
<td>70 p</td>
</tr>
<tr>
<td>78 x</td>
</tr>
</tbody>
</table>