CSE 30
Winter 2001
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. Leaf Subroutine ___________________  (20 points)

6. Local Variables, The Stack, and Return Values ___________________  (20 points)

7. SPARC Subroutines and Calling Convention ___________________  (15 points)

8. Floating Point ___________________  (12 points)

9. Linkage, Scope, Lifetime, Data ___________________  (31 points)

10. Machine Instructions ___________________  (20 points)

11. I/O & Instruction Speed ___________________  (8 points)

12. ALU & Control Unit ___________________  (8 points)

13. Miscellaneous ___________________  (12 points)

SubTotal ___________________  (203 points)

Extra Credit / C Tricks ___________________  (8 points)

Total ___________________  

Name _________________________
Signature_______________________
cs30x_________
1. Number Systems

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

octal ___________________________________

binary ___________________________________

decimal _________________________________

Convert $+277_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

sign-magnitude____________________________________________________

1’s complement____________________________________________________

2’s complement____________________________________________________

Convert $-194_{10}$ 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)

\[
\begin{array}{ccc}
11000101 & +00111011 & \hline \\
10111001 & +01010111 & \hline \\
00111001 & +01010110 & \hline \\
\end{array}
\]

\[
\begin{array}{ccccc}
\hline \\
| & | & | & | & | & | & | & | & | & | \\
\hline \\
\end{array}
\]

3. Branching

Fill in the SPARC assembly instructions to perform the following statements. **Do not optimize.** (16 points)

\[
\begin{array}{ccc}
_\text{C}_ & \text{SPARC assembly} & \text{int} \ x; \\
& & ! \ x \text{ mapped to } %10 \\
& & \hline \\
& & \text{for} ( x = 29; x > 21; --x ) \{ \\
& & \text{statement1;} \\
& & \text{if} ( x <= 25 ) \\
& & \text{statement2;} \\
& & \text{statement3;} \\
& & \} \\
\end{array}
\]

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

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>bcc</th>
<th>bpos</th>
<th>bvc</th>
<th>bleu</th>
<th>ble</th>
<th>bge</th>
<th>bneg</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addcc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.**

```plaintext
set  0x87EFAB34, %l0
set  0x43CDDC65, %l1
xor %l0, %l1, %l0

Value in %l0 is _______________________________________  (2 points)
```

```plaintext
set  0x87EFAB34, %l0
sra %l0, 10, %l0

Value in %l0 is _______________________________________  (2 points)
```

```plaintext
set  0x87EFAB34, %l0
sll %l0, 11, %l0

Value in %l0 is _______________________________________  (2 points)
```

```plaintext
set  0x56EFAB34, %l0
set  0xF1213121, %l1
bclr %l1, %l0

Value in %l0 is _______________________________________  (2 points)
```

```plaintext
set  0x56EFAB34, %l0
set  0x8A423121, %l1
btog %l1, %l0

Value in %l0 is _______________________________________  (2 points)
```
5. Leaf Subroutine

Write a full unoptimized SPARC assembly leaf subroutine to implement the checkRange() function from PA1. Return \textit{1} to indicate value \textit{is} in the range of \textit{minRange} - \textit{maxRange}; return \textit{0} to indicate value \textit{is not} in the range of \textit{minRange} - \textit{maxRange}. The ranges are inclusive. (20 points)

Function prototype for this function: \hspace{1em} \textbf{int checkRange( long value, long minRange, long maxRange );}
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 ) {

    int *local_stack_var1;
    struct foo {
        char s1;
        short s2[4];
        char s3;
        int s4;
    } local_stack_var2;

    local_stack_var2.s3 = 'F'; /* Use the ASCII value, not ‘F’ */
    *local_stack_var1++ = 4200;
    local_stack_var2.s2[2] = local_stack_var2.s2[3];

    return ( a + local_stack_var2.s4 );
}
```

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. (20 points)
7. SPARC Subroutines and Calling Convention (1 point each)

The ret instruction adds ________ to ________ and stores this result in ________.

The call instruction stores the current value of ________ into ________.

The C family of languages using the calling convention of call-by-___________.

Passing pointers is a way to simulate call-by-___________.

From a high-level language perspective, the two most important parts of a Stack/Call Frame are areas to store _________________ and _________________.

In the SPARC architecture, we pass the first ________ arguments to a function in %______ - %______.

Does a leaf subroutine have a Stack/Call Frame? ________

Does an open subroutine have a Stack/Call Frame? ________

Does a closed subroutine have a Stack/Call Frame? ________

8. Floating Point

Convert -54.375₁₀ (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 0x42C64000 (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;
int c = 20;
static int d;
static int
fubar( int e ) {
    float f = 420.420;
    int g;
    static int h = 20;
    static int *i;
    i = (int *) calloc( 3, sizeof(int) );
    ...
}
```

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 fubar().

a ______
b ______
c ______
d ______
e ______
f ______
g ______
h ______
i ______
fubar ______

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 fubar() is called to the point when fubar() returns.

a ______
b ______
c ______
d ______
e ______
f ______
g ______
h ______
i ______
fubar ______
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).

orcc %g2, -14, %o4 ___________________________________  (5 points)

bvs,a -16 instr. away ___________________________________  (5 points)

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

0xF42B0015 ___________________________________  (5 points)

0xAB3F600B ___________________________________  (5 points)

11. I/O & Instruction Speed

List the main advantage of having separate Instruction and Data L1 caches as opposed to both instructions and data sharing a single L1 cache. Assume size of I-cache + D-cache = size of a single shared cache. And assume the type of caches are the same (doesn’t matter what kind). (2 points)

Considering the following different SPARC assembly language instructions, specify whether the code in column A will execute slower/faster/same relative to the code in column B. (6 points)

Pick one of:

A slower than B
A faster than B
A same as B

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld [%l0], %l0</td>
<td>ld [%l1], %l1</td>
</tr>
<tr>
<td>add %l2, %l1, %l3</td>
<td>add %l2, %l1, %l3</td>
</tr>
<tr>
<td>mov 4000, %o3</td>
<td>set 5000, %o3</td>
</tr>
<tr>
<td>addcc %l1, %l2, %l3</td>
<td>addcc %l4, -5, %l5</td>
</tr>
</tbody>
</table>
12. ALU and Control Unit

Use the terms listed below to fill in the blanks. Terms may be used more than once. (1 point each)

The Control Unit gets its commands by ___________________ instructions from the ___________________.

The brains of the CPU is the ___________________ while the work is done by the ___________________.

which gets its operands from and stores its results in _________________.

_________________ are used by the Control Unit to tell the other part of the CPU what to do.

The ALU has direct access to the _________________ to read/write the _________________ with instructions like bne and addcc.

Tri-State device(s)  condition code bits
Control Unit L1 cache
Arithmetic Logic Unit (ALU) L2 cache
CPU Register(s) Translation Lookaside Buffer (TLB)
Program Counter (PC) Memory Management Unit (MMU)
Instruction Register (IR) input enable
Bus(es) output enable
Control Signal(s)/Line(s) combinational logic circuit/gate(s)
Process Status Register (PSR) sequential logic circuit/gate(s)
pipeline/pipelining synchronous/synchronizing
decode/decoding asynchronous/asynchronizing
encode/encoding

13. Miscellaneous

List two advantages of Dynamic Linking over Static 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. assembler 5. source code
2. linker 6. loader
3. compiler 7. preprocessor
4. program execution 8. executable (.exe/a.out)

_____ --> _____ --> _____ --> _____ --> _____ --> _____ --> _____ --> _____
Extra Credit / C Tricks

Here is a stripped-down version of a program I wrote to print out the hex values of single precision floating point values and to print the single precision floating point value of hex values to double-check my floating point conversions. For example, if I run the program on the command line as

```
a.out -5.5 0xC0B00000
```

the output is

```
-5.5 = 0xC0B00000
0xC0B00000 = -5.5
```

The last argument to the two printf() statements is missing. Write the appropriate expressions for these parameters. No other code is allowed — just expressions that would fit as a parameter expression. (8 points)

```c
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

int main( int argc, char *argv[] ) {
    float f;
    unsigned long l;
    int i;
    char *endptr;

    for ( i = 1; i < argc; ++i ) {
        if ( strchr( argv[i], '.' ) != NULL ) {
            errno = 0;
            f = strtod( argv[i], &endptr );
            if ( (errno == 0) && (*endptr == '\0') )
                (void) printf( "%s = 0x%08X\n", argv[i], _______________________________ );
        } else {
            errno = 0;
            l = strtoul( argv[i], &endptr, 16 );
            if ( (errno == 0) && (*endptr == '\0') )
                (void) printf( "%s = %g\n", argv[i], _______________________________ );
        }
    }
    return 0;
}
```
<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</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>48</td>
</tr>
<tr>
<td>50 P</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>68</td>
</tr>
<tr>
<td>70 p</td>
</tr>
<tr>
<td>78 x</td>
</tr>
</tbody>
</table>
Scratch Paper