CSE 30
Fall 2002
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 ___________________  (24 points)
7. SPARC Subroutines and Calling Convention ___________________  (18 points)
8. Floating Point ___________________  (12 points)
9. Machine Instructions ___________________  (20 points)
10. Linkage, Scope, Lifetime, Data ___________________  (32 points)
11. Load/Store/Memory ___________________  (9 points)
12. Miscellaneous ___________________  (28 points)

SubTotal ___________________  (211 points)
Extra Credit ___________________  (10 points)
Total ___________________
1. Number Systems

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

binary_______________________________________

octal _______________________________________

decimal _____________________________________

Convert $718_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

sign-magnitude_____________________________________

1’s complement_____________________________________

2’s complement_____________________________________ 

Convert $-279_{10}$ 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)

\[
\begin{array}{ccc}
10111001 & +10010110 & 10101011 \\
01010100 & +00111011 & +10101011 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
N & Z & V & C \\
\hline
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

3. Branching

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

```
/* %l2 mapped to local variable foo */
/* %l1 mapped to local variable bar */

mov  %l1, %l2
L1:
dec  %l2
cmp  %l2, 19
ble  L2
nop
add  %l2, 5, %l2
ba   L3
nop
L2:
sub  %l2, 5, %o0
call fubar
nop
mov  %o0, %l2
L3:
sub  %l2, 14, %l2
L4:
cmp  %l2, %g0
bge  L1
nop
```
4. Bit Operations

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

```
set 0xEBD43C9A, %l0
set 0x918A7161, %l1
and %l0, %l1, %l0
```

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

```
set 0xEBD43C9A, %l0
sra %l0, 9, %l0
```

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

```
set 0xEBD43C9A, %l0
sll %l0, 14, %l0
```

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

```
set 0xEBD43C9A, %l0
set __________, %l1
btog %l1, %l0 ! Value in %l0 is now OxFEEDFACE
```

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

```
set 0xEBD43C9A, %l0
set 0x918A7161, %l1
or %l0, %l1, %l0
```

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

```
set 0xEBD43C9A, %l0
srl %l0, 7, %l0
```

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

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

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

.section ".text"
main:
save  %sp, -92 & -8, %sp
mov   1375, %o0
call  recurse
nop
ret
restore

.global recurse  /* recurse.s */

[section ".rodata"
fmt:  .asciz  "%d "                       ! decimal # followed by a space

.section ".text"
recurse:
save  %sp, -(92 + 8) & -8, %sp           ! int a, result;
st   %g0, [%fp - 4]
st   %g0, [%fp - 8]

mov   %i0, %o0
mov   10, %o1
call  .rem
nop
st   %o0, [%fp - 4]

mov   %i0, %o0
mov   10, %o1
call  .div
nop
mov   %o0, %i0

cmp   %i0, %g0
be    over
nop

mov   %i0, %o0
call  recurse
nop

st   %o0, [%fp - 8]
over:
ld    [%fp - 4], %o0
ld    [%fp - 8], %o1
add   %o0, %o1, %o0
st    %o0, [%fp - 8]

set   fmt, %o0
ld    [%fp - 8], %o1
call  printf
nop

ld    [%fp - 8], %i0
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( char x, long y ) {
    short  local_stack_var1;
    struct foo {
        char            s1;
        int             s2;
        unsigned short  s3[3];
        long            s4;
    }      local_stack_var2;
    long  *local_stack_var3;
    local_stack_var2.s4 = --*local_stack_var3; /* 1 */
    x = local_stack_var2.s1;       /* 2 */
    local_stack_var1 = local_stack_var2.s3[1]++; /* 3 */
    return ( local_stack_var2.s2 + y ); /* 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.** (24 points)
7. SPARC Subroutines and Calling Convention (1 point each)

_________ subroutine is a subroutine that is expanded inline or is defined as a macro.

_________ subroutine is a subroutine that does not call other subroutines.

_________ subroutine is a traditional subroutine that passes parameters on the stack.

_________ subroutine is a subroutine that supports recursion.

_________ subroutine is a subroutine that generally increases generated code size.

_________ subroutine is a subroutine that does not guarantee its args will only be evaluated once.

_________ subroutine is a subroutine that uses save and restore instructions.

_________ subroutine is the type of subroutine of which .mul is an example.

_________ instruction slides the register set window down 16 registers.

_________ instruction saves the current value of %pc in %o7.

_________ instruction slides the register set window up 16 registers.

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

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

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

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

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

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

The _____________ function calls the instruction to set the program counter with the saved return address.
8. Floating Point

Convert \(-152.875\) \(_{10}\) (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 0xC2E24000 (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).

or \( %g0, %l2, %o1 \) ___________________________________  (5 points)
ldsb [\%o3], \%i5 ___________________________________  (5 points)

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

0x98870014 ___________________________________  (5 points)
0x26BFFFFFB ___________________________________  (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: (32 points — 1 point each)

```c
static int a = 701;          ____________
int b = 404;                  ____________
int c;                       ____________
static int d;                 ____________
static int foo( int e ) {     ____________ (foo) ____________ (e)
    static double f = 4.20;    ____________
    int g = 420;               ____________
    int (*h)(int) = foo;       ____________ (h) ____________ (where h is pointing)
    static int *i;
    i = (int *) malloc( g );  ____________ (where i is pointing)
...
}
```

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().

da ______
b ______
c ______
d ______
e ______
f ______
g ______
h ______
i ______
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.
a ______
b ______
c ______
d ______
e ______
f ______
g ______
h ______
i ______
foo ______
11. Load/Store/Memory

What gets printed in the following program? (9 points)

```assembly
.global main

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

c:     .byte 0xFF

.salign 2
s:     .half 0x1111

.salign 4
i1:    .word 0x98765432
i2:    .word 0x98765432
i3:    .word 0x98765432

.x:     .word 0xABCD

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

.set    i1, %l0

.set    s, %l1
.lduh   [%l1], %l1
.sth   %l1, [%l0+2]

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

.set    i2, %l0

.set    c, %l1
.ldb    [%l1], %l1
.stb   %l1, [%l0+1]

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

.set    x, %l0
.set    i3, %l1

.ldsh   [%l1], %l1
.st    %l1, [%l0]

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

ret
restore
```
12. Miscellaneous

What is the default buffering for (1 pt each)

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

Draw the logic circuit to perform the following boolean logic. Label each gate. (4 pts)

\[(a \land b) \land (\lnot a \oplus b)\]

Is this an example of a combinational logic circuit or a sequential logic circuit? (Circle correct answer.)

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 */
static int a = 911;
int main( int argc, char *argv[] ) {
    int b = 420;
    int c;
    foo( argc, b );
    /* 1 */ (void) printf( "argc --> %p\n", &argc );
    /* 2 */ (void) printf( "foo --> %p\n", foo );
    /* 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 = 404;
    static int g;
    /* 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 Credit
What gets printed? (5 points) Then rewrite the code take advantage of various optimizations that can be applied. Some optimizations are better/more complicated than others and are therefore worth more points. (5 points)

```
.global main

.code: .asciz "01001011"
fmt: .asciz "%s = %d\n"

main:
save %sp, -96, %sp       ! Optimize only the code below the save instr.
set code, %l0
ldub [%l0], %l1
clr %l2
ba test
nop

loop:
sub %l1, 0x30, %l1
mov %l2, %o0
mov 2, %o1
call .mul
nop

add %o0, %l1, %l2
inc %l0
ldub [%l0], %l1

test:
cmp %l1, %g0
bne loop
nop

set fmt, %o0
set code, %o1
mov %l2, %o2
call printf
nop

ret
restore
```

Output: _______________________________ (5 points)
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>7D )</td>
<td>7E ~</td>
<td>7F DEL</td>
</tr>
</tbody>
</table>
Scratch Paper