CSE 30
Winter 2003
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 ___________________  (29 points)

SubTotal ___________________  (212 points)
Extra Credit ___________________  (10 points)
Total ___________________
1. Number Systems

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

binary_______________________________________

octal _______________________________________________________________________
decimal ___________________________________________________________________

Convert $736_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

sign-magnitude_______________________________________________

1’s complement_______________________________________________

2’s complement_______________________________________________

Convert $-227_{10}$ 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)

\[
\begin{array}{ccc}
00111011 & +10111001 & \\
11010110 & +11010100 & \\
01010101 & +10101011 & \\
\hline
\end{array}
\]

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

3. Branching

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

\[
\text{SPARC assembly}
\]

mov %l2, %l1 /* %l2 mapped to local variable foo */

L1:

inc %l1

cmp %l1, 7

bl L2

nop

add %l2, 19, %o0

call fubar

nop

mov %o0, %l1

ba L3

nop

L2:

sub %l1, 6, %l1

L3:

add %l1, 7, %l1

L4:

cmp %l1, %g0

bge L1

nop
4. Bit Operations

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

set 0x917A7164, %l0
set 0xDB3CB357, %l1
and %l0, %l1, %l0

Value in %l0 is ________________________________ (2 points)

set 0x917A7164, %l0
sra %l0, 14, %l0

Value in %l0 is ________________________________ (2 points)

set 0x917A7164, %l0
sll %l0, 7, %l0

Value in %l0 is ________________________________ (2 points)

set 0x917A7164, %l0
set 0x?????????, %l1
btog %l1, %l0

Value in %l0 is now 0xFACEFEED
Value set in %l1 must be this bit pattern ________________________________ (3 points)

set 0x917A7164, %l0
set 0xDB3CB357, %l1
or %l0, %l1, %l0

Value in %l0 is ________________________________ (2 points)

set 0x917A7164, %l0
srl %l0, 11, %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  "%c"                      ! character format specifier

.section "".text"
recurse:
save   %sp, -(92 + 8) & -8, %sp        ! int mod, div;
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
st     %o0, [%fp - 8]
set     fmt, %o0
ld     [%fp - 4], %i0
add    %i0, 0x30, %o1
call   printf ____________________________________
nop
ld     [%fp - 8], %o0
cmp    %o0, %g0
be     over
nop
ld     [%fp - 8], %o0
call   recurse
nop
over:
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
#include <stdio.h>

int fubar( long x, char y ) {
    short  local_stack_var1;
    struct foo {
        int    s1;
        short  s2[3];
        char   s3;
        long   s4;
    } local_stack_var2;
    long  *local_stack_var3;

    y = local_stack_var2.s3;          /* 1 */
    local_stack_var3++;              /* 2 */
    local_stack_var1 = local_stack_var2.s2[2] + 15; /* 3 */

    return ( local_stack_var2.s4 + x ); /* 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

Write the entire source module for function add3() as a leaf subroutine in SPARC Assembly. (6 points)

```c
int add3( int a, int b, int c ); /* Function Prototype */
/* Add a + b + c and return this sum. */
```

**SPARC Assembly**

Write the entire source module for function mult3() as an unoptimized closed subroutine in SPARC Assembly. (12 points)

```c
long mult3( int a, int b, int c ); /* Function Prototype */
/* Multiply a * b * c and return this product. */
```

**SPARC Assembly**
8. Floating Point

Convert -139.125_{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 0x42E2C000 (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).

orc  %l4, 15, %g2 ___________________________________  (5 points)
sth  %i4, [%o2 + %l3] ___________________________________  (5 points)

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

0x9DE3BFA0 ___________________________________ (5 points)
0x36BFFFFC ___________________________________ (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; __________________________
int b; __________________________
int c = 101; __________________________
static int d = 404; __________________________
int foo( int e ) { __________________________ (foo) __________________________ (e)
  double f = 4.20; __________________________
  static int g = 420; __________________________
  static int *h; __________________________
  h = (int *) malloc( b ); __________________________ (where h is pointing)
  int (*i)(int) = foo; __________________________ (i) __________________________ (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().

a _______
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 0XXXXXXXX

c: .byte 0x88

.s: .half 0xABCD

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

x: .word 0xFFFF

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

set i1, %li0
set s, %li1
ldsh [%li1], %li1
stb %li1, [%li0+3]

set fmt, %lo0
ld [%li0], %lo1
call printf _________________________________
nop

set i2, %li0
set c, %li1
ldsb [%li1], %li1
stb %li1, [%li0]

set fmt, %lo0
ld [%li0], %lo1
call printf _________________________________
nop

set x, %li0
set i3, %li1

ld [%li1], %li1
stb %li1, [%li0]

set fmt, %lo0
ld [%li0], %lo1
call printf _________________________________
nop

ret
restore
```
12. Miscellaneous

Which is faster? Circle correct answer. (1 pt each)

- Programmed I/O or Interrupt-Driven I/O
- L1 cache or Buffer cache in main memory
- Writes to Write-Through or Write-Back cache
- Getting back on the CPU after time quantum expires or after initializing an I/O operation

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

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

Why do computer programmers confuse Halloween with Christmas? (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[] ) {
    int b;
    int c = 420;
    foo( argc, b );
    /* prints smallest value */
    /* 1 */ (void) printf( "c --> %p\n", &c );
    /* 2 */ (void) printf( "b --> %p\n", &b );
    /* 3 */ (void) printf( "a --> %p\n", &a );
    /* 4 */ (void) printf( "malloc --> %p\n", malloc(50) );
    /* 5 */ (void) printf( "argc --> %p\n", &argc );
    /* 6 */ (void) printf( "foo --> %p\n", foo );
}

void foo( int d, int e ) {
    static int f = 404;
    int g;
    /* prints */
    /* 7 */ (void) printf( "g --> %p\n", &g );
    /* 8 */ (void) printf( "f --> %p\n", &f );
    /* 9 */ (void) printf( "e --> %p\n", &e );
    /* 10 */ (void) printf( "d --> %p\n", &d );
}
```
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 "111"
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:
        mov %l2, %o0
        mov 16, %o1
        call .mul
        nop
        sub %l1, 0x30, %l1
        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)
<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>
Scratch Paper
Scratch Paper