CSE 30
Fall 2005
Final Exam

1. Number Systems
   ________ (15 points)

2. Binary Addition/Condition Code Bits/Overflow Detection
   ________ (12 points)

3. Branching
   ________ (22 points)

4. Bit Operations
   ________ (13 points)

5. Recursion/SPARC Assembly
   ________ (10 points)

6. Local Variables, The Stack, Return Values
   ________ (24 points)

7. SPARC Leaf Subroutines
   ________ (23 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
    ________ (33 points)

SubTotal
   ________ (225 points)

Extra Credit
   ________ (10 points)

Total
   ________
1. Number Systems

Convert $\text{FB8E}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

binary _______________________________________

octal _______________________________________

decimal _______________________________________

Convert $274_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

sign-magnitude_________________________________

1’s complement_________________________________

2’s complement_________________________________

Convert $-695_{10}$ 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)

```
  01010101  11010110  00111011
+10101011  +11010100  +10111001
---------  ---------  ---------
N  Z  V  C  N  Z  V  C  N  Z  V  C
|   |   |   |   |  |   |   |   |  |   |   |   |
---------  ---------  ---------
```

3. Branching

Write the SPARC assembly statements to perform the following C statements. Do not optimize. (22 points)

C
```c
int x;  
int y;  
for ( x = 75; x >= 15; x = x - 2) {
    y = bar( x );
    if ( y <= x )
        x = x % 5;
}
```
4. Bit Operations

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

```plaintext
set 0xFADED420, %l0
set 0x420FADED, %l1
or  %l0, %l1, %l0

Value in %l0 is ____________________________ (2 points)

set 0xFADED420, %l0
sll  %l0, 14, %l0

Value in %l0 is ____________________________ (2 points)

set 0xFADED420, %l0
srl  %l0, 7, %l0

Value in %l0 is ____________________________ (2 points)

set 0xFADED420, %l0
set 0x????????, %l1
xor  %l0, %l1, %l0 ! Value in %l0 is now 0xBABEFEED

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

set 0xFADED420, %l0
set 0x420FADED, %l1
and  %l0, %l1, %l0

Value in %l0 is ____________________________ (2 points)

set 0xFADED420, %l0
sra  %l0, 9, %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 */

.main:
    save %sp, -92 & -8, %sp
    set 1948247651, %o0
    call recurse
    nop
    ret
    restore

.global recurse
/* recurse.s */

.fmt: .asciz "%d\n" ! decimal # followed by a newline

.recurse:
    save %sp, -(92 + 8) & -8, %sp ! 2 local int variables
clr %10
-hop:
    st %g0, [%fp - 4]
    st %g0, [%fp - 8]
    mov %10, %o0
    mov 10, %o1
    call .rem
    nop
    mov %o0, [%fp - 4]
    mov %10, %o0
    mov 10, %o1
    call .div
    nop
    mov %o0, %10
    inc %10
    andcc %10, 0x01, %g0
    bne hop
    nop
    cmp %10, %g0
    be whoa
    nop
    mov %10, %o0
    call recurse
    nop
    st %o0, [%fp - 8]

.whoa:
    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], %10
    ret
    restore
```

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

    local_stack_var1 = local_stack_var2.s1[1] + 13; /* 1 */
    local_stack_var2.s2 = y; /* 2 */
    local_stack_var3 = &local_stack_var2.s4; /* 3 */

    return ( x + *local_stack_var3++ ); /* 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 Leaf Subroutines

Write a full unoptimized leaf SPARC assembly function translation of the following C function to determine how many odd numbered bits in the parameter value are set. Return the number of odd numbered bits that are set to 1. The most significant bit is numbered bit 31 (odd); the least significant bit is numbered bit 0 (even). Be sure to state which registers you are using for the various local variables and parameters. (23 pts)

For example,  \( \text{countOddBitsSet}( 0xAAAAAAAA ) \) will return 16
\( \text{countOddBitsSet}( 0xA08AA85A ) \) will return 10

\[
\begin{align*}
\text{C} & \quad \text{Leaf SPARC Assembly Subroutine} \\
\text{int countOddBitsSet( unsigned int value )} \{ \\
\text{unsigned int mask = 2;} \\
\text{int i;} \\
\text{int cnt = 0;} \\
\text{for ( i = 0; i < 16; ++i )} \{ \\
\text{if ( (value & mask) != 0 )} \\
\text{++cnt;} \\
\text{mask = mask << 2;} \\
\} \\
\text{return cnt;} \\
\}
\end{align*}
\]
8. Floating Point

Convert \(142.875_{10}\) (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary fixed-point</td>
<td>_______________ (2 points)</td>
</tr>
<tr>
<td>IEEE floating-point</td>
<td>_______________ (4 points)</td>
</tr>
</tbody>
</table>

Convert \(0xC344C000\) (single-precision IEEE floating-point representation) to fixed-point decimal.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-point decimal</td>
<td>_______________ (6 points)</td>
</tr>
</tbody>
</table>

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).

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>STB</td>
<td>%l3, [%o1 + %i6] _______________ (5 points)</td>
</tr>
<tr>
<td>ADDCC</td>
<td>%o2, %l2, %i5 _______________ (5 points)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xD43D001B</td>
<td>_______________ (5 points)</td>
</tr>
<tr>
<td>0x9A18FFF6</td>
<td>_______________ (5 points)</td>
</tr>
</tbody>
</table>
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
int a = 411; ______________
static int b; ______________
static int c = 404; ______________
int d; ______________
int foo( int e ) {
    static double f = 4.20; ______________
    static int g; ______________
    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)

.global main

.save %sp, -96, %sp
.set i1, %10
.set s, %11
.ldsh [%11], %11
.st %11, [%10]
.stb %11, [%10+1]

.set fmt, %o0
.ld [%10], %o1
.call printf ________________________
.nop

.set i2, %10
.set c, %11
.ldsb [%11], %11
.sth %11, [%10+2]
.stb %11, [%10+1]

.set fmt, %o0
.ld [%10], %o1
.call printf ________________________
.nop

.set x, %10
.set i3, %11
.ldsb [%11+2], %12
.sth %12, [%10]
.stb %12, [%10+3]

.set fmt, %o0
.ld [%10], %o1
.call printf ________________________
.nop

.ret
.restore
12. Miscellaneous

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

1. program execution 2. compiler 3. executable (.exe/a.out)
7. linker/linkage editor 8. loader

_____ -> _____ -> _____ -> _____ -> _____ -> _____ -> _____ -> _____

Draw the logic circuit to perform the following boolean logic. Label each gate. Do not optimize. (3 pts)

(a & b) | (!a ^ b)

Is this a sequential or combinational logic circuit? (Circle correct answer in the question to the left.) (1 pt)

What is the name of the Martini Bar in Pacific Beach that has Computer Trivia Night? (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 = 404;

int main( int argc, char *argv[] ) {
    int b;
    static int c;

    foo( argc, b );

    /* 1 */ (void) printf( "c --> %p\n", &c );
    /* 2 */ (void) printf( "argc --> %p\n", &argc );
    /* 3 */ (void) printf( "foo --> %p\n", foo );
    /* 4 */ (void) printf( "malloc --> %p\n", malloc(50) );
    /* 5 */ (void) printf( "b --> %p\n", &b );
}

void foo( int d, int e ) {
    int f = 911;
    int g;

    /* 6 */ (void) printf( "g --> %p\n", &g );
    /* 7 */ (void) printf( "e --> %p\n", &e );
    /* 8 */ (void) printf( "f --> %p\n", &f );
    /* 9 */ (void) printf( "d --> %p\n", &d );
    /* 10 */ (void) printf( "a --> %p\n", &a );
}
```
Extra Credit
What does the following SPARC assembly language program output?

```
.global main

.section ".rodata"
fmt: .asciz "%c"
.align 2
foo: .half 0x4500, 0x4264, 0x5565, 0x336B, 0x4261, 0x204E
     .half 0x4220, 0x3266, 0x3472, 0x3075, 0x2153, 0x0000
bar: .half 0x0000

.section ".text"

main: main:
    save %sp, -96, %sp
set     foo, %l0
clr     %l1
clr     %l2
clr     %l3
add     %l3, %l2, %l4
ldub    [%l0+%l4], %o1
tst     %o1
be      L1
nop
L2:
    set     fmt, %o0
ldub    [%l0+%l4], %o1
call    printf
nop
inc     %l1
mov     %l1, %o0
mov     2, %o1
call    .mul
nop
mov     %o0, %l2
add     %l3, %l2, %l4
ldub    [%l0+%l4], %o1
tst     %o1
bne     L2
nop
L1:
    set     fmt, %o0
mov     0x0A, %o1
call    printf
nop
    ret
    restore
```

Output ________________________ (2 points)
Now optimize the code to get the same result with the fewest cycles. Some optimizations are better than others. You may not be able to eliminate all nops. Go for the fewest machine cycles assuming memory accesses are several more cycles than other non-memory access instructions. You cannot change the overall algorithm.

(8 points)

```
set   value, reg.rd
expands to
sethi  %hi(value), reg.rd
or    reg.rd, %lo(value), reg.rd
```
<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>