CSE 30  
Fall 2008  
Final Exam

1. Number Systems  
2. Binary Addition/Condition Code Bits/Overflow Detection  
3. Branching  
4. Bit Operations  
5. Recursion/SPARC Assembly  
6. Local Variables, The Stack, Return Values  
7. More Recursive Subroutines  
8. Floating Point  
9. Machine Instructions  
10. Linkage, Scope, Lifetime, Data  
11. Load/Store/Memory  
12. Miscellaneous

SubTotal  
Extra Credit  
Total

______________________  (15 points)
______________________  (12 points)
______________________  (22 points)
______________________  (13 points)
______________________  (10 points)
______________________  (19 points)
______________________  (16 points)
______________________  (12 points)
______________________  (20 points)
______________________  (32 points)
______________________  (11 points)
______________________  (27 points)
______________________  (209 points)
______________________  (11 points)
______________________
1. Number Systems

Convert **0xFADE** (2’s complement, 16-bit word) to the following. (6 points)

- **binary** ___________________________ (straight base conversion)
- **octal** ___________________________ (straight base conversion)
- **decimal** ___________________________ (convert to signed decimal)

Convert **298** to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

- **sign-magnitude** ___________________________
- **1’s complement** ___________________________
- **2’s complement** ___________________________

Convert **-542** 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)

- 10101010 +11010110
- 01110110 +11001101
- 11111111 +00000001

<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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Branching
Write the SPARC Assembly leaf subroutine to perform the following C function. Use only standard looping/branching format as detailed in class and class notes. Comment! **Do not optimize.** (22 points)

```c
int checkIfPowerOf2( unsigned int value )
{
    int i;
    int cnt = 0;
    unsigned int mask = 0x80000000;
    for ( i = 0; i < 32; ++i )
    {
        if ( (value & mask) != 0 )
            ++cnt;
        if ( cnt > 1 )
            return 0;
        mask = mask >> 1;
    }
    return cnt;
}
```

```sparc-assembler
.global checkIfPowerOf2  ! i  mapped to %o1
.section " .text"  ! cnt mapped to %o3
checkIfPowerOf2:  ! mask mapped to %o5
    int i;
    int cnt = 0;
    unsigned int mask = 0x80000000;
    for ( i = 0; i < 32; ++i )
    {
        if ( (value & mask) != 0 )
            ++cnt;
        if ( cnt > 1 )
            return 0;
        mask = mask >> 1;
    }
    return cnt;
```
4. Bit Operations

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

```
set 0xCA4273BD, %l0
set 0x9035768A, %l1
and %l0, %l1, %l0

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

```
set 0xCA4273BD, %l0
sra %l0, 13, %l0

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

```
set 0xCA4273BD, %l0
sll %l0, 9, %l0

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

```
set 0xCA4273BD, %l0
set 0x????????, %l1
xor %l0, %l1, %l0

! Value in %l0 is now OxFEEDBEEF

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

```
set 0xCA4273BD, %l0
set 0x9035768A, %l1
or %l0, %l1, %l0

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

```
set 0xCA4273BD, %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, -96, %sp
set 24568, %o0
call recurse
nop
ret
restore

.global recurse
/* recurse.s */

.section ".rodata"
fmt: .asciz "%d\n"

.section ".text"
recurse:
save %sp, -(92 + 4) & -8, %sp ! One int local variable on the stack
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
set fmt, %o0
ld [%fp - 4], %o1
call printf
nop
cmp %i0, %g0
be base
nop
mov %i0, %o0
call recurse
nop
base:
set fmt, %o0
mov %i0, %o1
call printf
nop
ret
restore
```
6. Local Variables, The Stack, and Return Values
Here is a C function that allocates a few 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( short x, long y ) {
    char  *local_stack_var1;
    short *local_stack_var2;
    struct foo {
        short s1;
        char s2[3];
        short s3[3];
        int s4;
    } local_stack_var3;
    local_stack_var1 = &local_stack_var3.s2[2]; /* 1 */
    local_stack_var2 = local_stack_var3.s3 + y; /* 2 */
    local_stack_var3.s1 = y + *++local_stack_var2; /* 3 */
    return ( local_stack_var3.s4 + x ); /* 4 */
}
```

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. (19 points)
#include <stdio.h>

int mystery1( int x )
{
    int result;

    printf( "x = %d\n", x );

    if ( x <= 0 )
        return 0;
    else {
        result = x + mystery2( x - 1 );
        printf( "result = %d\n", result );
        return result;
    }
}

int mystery2( int x )
{
    int result;

    printf( "x = %d\n", x );

    if ( x <= 0 )
        return 0;
    else {
        result = x + mystery1( x - 2 );
        printf( "result = %d\n", result );
        return result;
    }
}

int main( int argc, char *argv[] )
{
    printf( "%d\n", mystery1( 10 ) );
    return 0;
}
8. Floating Point

Convert $129.125_{10}$ (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 $0xC348C000$ (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).

and   %i2, %l3, %o4 ___________________________________  (5 points)
std   %o2, [%fp - 8] ___________________________________  (5 points)

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

0x3CBFFFF9 ___________________________________  (5 points)
0xEC0B401C ___________________________________  (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)

static int a = 411; ______________
int b; ______________
int c = 404; ______________
static int d; ______________
int foo( int e ) { ____________ (foo) ______________ (e)
double f = 4.20; ______________
static int g = 8675309; ______________
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.
11. Load/Store/Memory
What gets printed in the following program? (11 points)

```assembly
.global main

.section "".data"
fmt: .asciz "0x%08X\n" ! prints value as hex 0xXXXXXXXX

.c: .byte 0x83

.s: .half 0xBEAD

.i1: .word 0xABCD1234
.i2: .word 0xABCD1234
.i3: .word 0xABCD1234
.x: .word 0x5FF50000

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

set x, %l0
set s, %l1
lduh [%l1], %l2  _____________________ Hex value in %l2
stb %l2, [%l0+2]  ______________________ Hex value in word labeled x
sll %l2, 4, %l2  _____________________ Hex value in %l2
stb %l2, [%l0+1]

set fmt, %o0
ld [%l0], %o1
call printf  _____________________ Hex value in word labeled x
nop (same as output of this printf)

set i1, %l0
set c, %l1
ldsb [%l1], %l2  _____________________ Hex value in %l2
stb %l2, [%l0+2]  ______________________ Hex value in word labeled i1
stb %l2, [%l0+1]

set fmt, %o0
ld [%l0], %o1
call printf  _____________________ Hex value in word labeled i1
nop (same as output of this printf)

set i2, %l0
set i3, %l1
ld [%l1], %l2  _____________________ Hex value in %l2
stb %l2, [%l0]  ______________________ Hex value in word labeled i2
sra %l2, 8, %l2  _____________________ Hex value in %l2
stb %l2, [%l0+2]

set fmt, %o0
ld [%l0], %o1
call printf  _____________________ Hex value in word labeled i2
nop (same as output of this printf)

ret
restore
```
12. Miscellaneous

Fill in the blanks so the following program correctly determines if it is run on a Big-Endian or Little-Endian architecture. (2 points)

```c
#include <stdio.h>

int main( void ) {
    int word = 1;
    if ( *(char *)&word == 1 )
        printf( "____________________________\n" ); /* Prints either Big-Endian or Little-Endian */
    else
        printf( "____________________________\n" ); /* Prints either Big-Endian or Little-Endian */
    return 0;
}
```

What gets printed with the statements below? (4 points)

```c
unsigned short x = 0xF024;
putchar( (x & 0xF) + '0' );     ________
putchar( (x << 8 >> 12) + '0' );  ________
putchar( (x & 0xF00) + '0' );  ________
putchar( (x >> 12) + '0' );  ________
```

What is Rick's favorite TV show? (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[] ) {
    static int b = 311;
    int c = 69;

    foo( argc, b );
    /* 1 */ (void) printf( "argc --> %p\n", &argc );
    /* 2 */ (void) printf( "foo --> %p\n", foo );
    /* 3 */ (void) printf( "a --> %p\n", &a );
    /* 4 */ (void) printf( "malloc --> %p\n", malloc(50) );
    /* 5 */ (void) printf( "c --> %p\n", &c );
    /* 6 */ (void) printf( "b --> %p\n", &b );
}

void foo( int d, int e ) {
    int f = e;
    int g = f;
    /* 7 */ (void) printf( "e --> %p\n", &e );
    /* 8 */ (void) printf( "g --> %p\n", &g );
    /* 9 */ (void) printf( "d --> %p\n", &d );
    /* 10 */ (void) printf( "f --> %p\n", &f );
}
```
**Extra Credit** (11 points)

What is the value of each of the following expressions taken sequentially based on changes that may have been made in previous statements?

```c
char a[] = "SD Chargers!";
char *p = a + 3;

printf( "%c", --*p );
++p;
printf( "%c", *p = *p + 4 );
printf( "%c", p[1] = *(a + strlen(a) - 2) + 2 );
p = p + 2;
printf( "%c", *p = p[-2] + 2 );
p++;
printf( "%c", *p = p[0] - 3 );
printf( "%d", ++p - a );
printf( "\n%s\n", a );
```

Optimize the following assembly code fragments using only the given instructions. Some optimizations are worth more than others.

```
mov    %l3, %o0
mov    32, %o1
call   .mul
nop
mov    %o0, %l3
```

```
cmp    %l1, %i2
bge    L1
nop
add    %l0, %l1, %i2
L1:
andcc  %o0, %i2, %o2
sub    %o2, 5, %o2
```
Hexadecimal - Character

| 00 NUL | 01 SOH | 02 STX | 03 ETX | 04 EOT | 05 ENQ | 06 ACK | 07 BEL |
| 08 BS  | 09 HT  | 0A NL  | 0B VT  | 0C NP  | 0D CR  | 0E SO  | 0F SI  |
| 10 DLE | 11 DC1 | 12 DC2 | 13 DC3 | 14 DC4 | 15 NAK | 16 SYN | 17 ETB |
| 18 CAN | 19 EM  | 1A SUB | 1B ESC | 1C FS  | 1D GS  | 1E RS  | 1F US  |
| 20 SP  | 21 !   | 22 "   | 23 #   | 24 $   | 25 %   | 26 &   | 27 '   |
| 28 (   | 29 )   | 2A *   | 2B +   | 2C ,   | 2D -   | 2E .   | 2F /   |
| 30 0   | 31 1   | 32 2   | 33 3   | 34 4   | 35 5   | 36 6   | 37 7   |
| 38 8   | 39 9   | 3A :   | 3B ;   | 3C <   | 3D =   | 3E >   | 3F ?   |
| 40 @   | 41 A   | 42 B   | 43 C   | 44 D   | 45 E   | 46 F   | 47 G   |
| 48 H   | 49 I   | 4A J   | 4B K   | 4C L   | 4D M   | 4E N   | 4F O   |
| 50 P   | 51 Q   | 52 R   | 53 S   | 54 T   | 55 U   | 56 V   | 57 W   |
| 58 X   | 59 Y   | 5A Z   | 5B [   | 5C \   | 5D ]   | 5E ^   | 5F _   |
| 60 `   | 61 a   | 62 b   | 63 c   | 64 d   | 65 e   | 66 f   | 67 g   |
| 68 h   | 69 i   | 6A j   | 6B k   | 6C l   | 6D m   | 6E n   | 6F o   |
| 70 p   | 71 q   | 72 r   | 73 s   | 74 t   | 75 u   | 76 v   | 77 w   |
| 78 x   | 79 y   | 7A z   | 7B {   | 7C |   | 7D }   | 7E ~   | 7F DEL |

A portion of the Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ postfix increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- postfix decrement</td>
<td></td>
</tr>
<tr>
<td>* indirection</td>
<td>R to L</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td></td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td></td>
</tr>
<tr>
<td>&amp; address-of</td>
<td></td>
</tr>
<tr>
<td>* multiplication</td>
<td>L to R</td>
</tr>
<tr>
<td>/ division</td>
<td></td>
</tr>
<tr>
<td>% modulus</td>
<td></td>
</tr>
<tr>
<td>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>