# CSE 30
Fall 2006
Final Exam

<table>
<thead>
<tr>
<th>Section</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Number Systems</td>
<td>15</td>
</tr>
<tr>
<td>2. Binary Addition/Condition Code Bits/Overflow Detection</td>
<td>12</td>
</tr>
<tr>
<td>3. Branching</td>
<td>20</td>
</tr>
<tr>
<td>4. Bit Operations</td>
<td>13</td>
</tr>
<tr>
<td>5. Recursion/SPARC Assembly</td>
<td>10</td>
</tr>
<tr>
<td>6. Local Variables, The Stack and Return Values</td>
<td>27</td>
</tr>
<tr>
<td>7. Bit Slinging</td>
<td>12</td>
</tr>
<tr>
<td>8. Floating Point</td>
<td>12</td>
</tr>
<tr>
<td>9. Machine Instructions</td>
<td>20</td>
</tr>
<tr>
<td>10. Linkage, Scope, Lifetime, Data</td>
<td>34</td>
</tr>
<tr>
<td>11. Load/Store/Memory</td>
<td>9</td>
</tr>
<tr>
<td>12. Miscellaneous</td>
<td>36</td>
</tr>
</tbody>
</table>

**SubTotal**  
(220 points)

**Extra Credit**  
(10 points)

**Total**  

1. Number Systems
Convert 0xF8F3 (2’s complement, 16-bit word) to the following. (6 points)

- **binary**____________________________________ (straight bit pattern translation)
- **octal**______________________________________ (straight bit pattern translation)
- **decimal**____________________________________ (pos/neg decimal value from 2’s complement encoding)

Convert 329\textsubscript{10} to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

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

Convert -598\textsubscript{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)

<table>
<thead>
<tr>
<th>11010110</th>
<th>10111001</th>
<th>01011101</th>
</tr>
</thead>
<tbody>
<tr>
<td>+10110111</td>
<td>+01010110</td>
<td>+01101001</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
</tbody>
</table>

```
N  Z  V  C
| | | | |
| | | | |
| | | | |
N  Z  V  C
| | | | |
| | | | |
N  Z  V  C
| | | | |
```

2
3. Branching (20 points)
Given the following SPARC assembly code write the equivalent function in C. Use only standard loops and if/else statements; no gotos! Just perform a direct translation.

```
.globl branching
.sec ".text"
branching:
    save %sp, -96, %sp
    call rand
    nop
    mov %o0, %l1     ! local var int x
        ! mapped to %l1
    cmp %l1, 99
    bge L4
    nop
    L1:
    cmp %l1, 42
    bl L2
    nop
    add %l1, 4, %l1
    ba L3
    nop
    L2:
    mov %l1, %o0
    mov 5, %o1
    call .rem
    nop
    mov %o0, %l1
    L3:
    cmp %l1, 99
    bl L1
    nop
    L4:
    add %l1, 17, %i0
    ret
    restore
```

For the following instruction sequence, mark with an X the conditional branch instructions which would transfer control to doit if used in place of ba. [+1 correct; -1 incorrect]

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>blu</th>
<th>bpos</th>
<th>bgu</th>
<th>bne</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>cmp</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>
</tbody>
</table>

[+1 correct; -1 incorrect]
4. Bit Operations

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

```plaintext
set  0xAC2006ED, %l0
set  0x98675309, %l1
or  %l0, %l1, %l0

Value in %l0 is __________________________ (2 points)

set  0xAC2006ED, %l0
sll  %l0, 5, %l0

Value in %l0 is __________________________ (2 points)

set  0xAC2006ED, %l0
srl  %l0, 6, %l0

Value in %l0 is __________________________ (2 points)

set  0xAC2006ED, %l0
set  0x????????, %l1
xor  %l0, %l1, %l0
 ! Value in %l0 is now OxDEADBEEF

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

set  0xAC2006ED, %l0
set  0x98675309, %l1
and  %l0, %l1, %l0

Value in %l0 is __________________________ (2 points)

set  0xAC2006ED, %l0
sra  %l0, 13, %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

.section ".rodata"
.align 4
code: .word 0x65756F50, 0x72676146, 0x00336E4A, 0x306F2020, 0x4C3E5220, 0x20756543
.word 0x6C760065, 0x6900734C, 0x00000030

.main:
save %sp, -96, %sp
set code, %o0
mov 0, %o1
call fubar
nop
ret
restore

.global fubar

.section ".rodata"
fmt: .asciz "%c"

.section ".text"
fubar:
save %sp, -(92 + 1) & -8, %sp
andcc %i1, 0x01, %g0
bne L1
nop
add %i1, 3, %l0
ba L2
nop
L1:
add %i1, 5, %l0
L2:
and %l0, 0x07, %l0
ldub [%i0 + %l0], %l1
stb %l1, [%fp - 1]
cmp %l0, 0x04
be end
nop
add %i1, 1, %l0
mov %l0, %o0
call fubar
nop
end:
set fmt, %o0
ldub [%fp - 1], %o1
call printf
nop
ret
restore
```

Output ____________________________
6. Local Variables, The Stack, and Return Values
Here is a C function that allocates a couple local variables, performs some assignments, returns a value. Don’t worry about any local variables not being initialized before being used. Just do a direct translation. **Draw lines.**

```c
long fubar( char *x, int y ) {
    int  *local_stack_var1;
    struct foo {
        short s1[5];
        char  s2;
        char *s3;
        int   s4;
    }     local_stack_var2;
    short local_stack_var3;

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

    return ( y + *local_stack_var1++ ); /* 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.** (27 points)
7. Bit Slinging and Leaf Subroutines

Write a full unoptimized leaf SPARC assembly function translation of the following C function to determine whether an unsigned value is a power of 2. Return 1 to indicate true; return 0 to indicate false. Be sure to state which registers you are using for the various local variables and parameters. (23 pts)

For example,

- checkIfPowerOf2( 2048 ) will return 1
- checkIfPowerOf2( 6 ) will return 0
- checkIfPowerOf2( 3048 ) will return 0
- checkIfPowerOf2( 16 ) will return 1

C

```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;
        mask = mask >> 1;
    }

    if ( cnt == 1 )
        return 1;
    else
        return 0;
}
```

Leaf SPARC Assembly Subroutine
8. Floating Point

Convert -154.125_{10} (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary fixed-point</td>
<td>______________</td>
</tr>
<tr>
<td>IEEE floating-point</td>
<td>______________</td>
</tr>
</tbody>
</table>

Convert 0x434FE000 (single-precision IEEE floating-point representation) to fixed-point decimal.

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed-point decimal</td>
<td>______________</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>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sth %i1, [%l2 + %o3]</code></td>
<td>______________</td>
</tr>
<tr>
<td><code>subcc %l2, -9, %l5</code></td>
<td>______________</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Machine Code Instruction</th>
<th>Assembly Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xE64FBFFB</td>
<td>______________</td>
</tr>
<tr>
<td>0x2E800010</td>
<td>______________</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: (34 points — 1 point each)

```c
static int a = 411; ______________
int b; ______________
static int c; ______________
int d = 420;

static int
foo( int e ) {
    ____________ (foo) ____________ (e)
    static double f;
    ______________
    int *g;
    ______________
    static int h = 911;
    ______________
    int (*i)(int) = foo; ____________ (i) ____________ (where i is pointing)
    g = (int *) malloc( d ); ______________ (where g 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) 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 _______ a _______
b _______ b _______
c _______ c _______
d _______ d _______
e _______ e _______
f _______ f _______
g _______ g _______
h _______ h _______
i _______ i _______
foo _______ foo _______

If function foo() is called 7 times, how many times does the variable h get initialized to 911? ____________
In function foo(), what is the initial value of g? ______________

11. Load/Store/Memory
What gets printed in the following program? (9 points)

```assembly
.global main

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

c:      .byte   0x9A

.salign 2
s:      .half   0xDCBA

.salign 4
i1:     .word   0x935ACDC9
i2:     .word   0x935ACDC9
i3:     .word   0x935ACDC9
x:      .word   0x00005678

.section ".text"
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
    ldub  [%11], %12
    stb   %12, [%10]
    sth   %12, [%10+2]
    set   fmt, %o0
    ld    [%10], %o1
    call  printf

nop
    set   x, %10
    set   i3, %11
    ldsb  [%11+1], %12
    sth   %12, [%10]
    stb   %12, [%10+3]
    set   fmt, %o0
    ld    [%10], %o1
    call  printf

nop

ret```
12. Miscellaneous

Put the following in the correct order/sequence using the numbers to the left of each word. (11 pts)


What is the default buffering for each of the following. (4 pts)

stdin _________________ stderr _________________
stdout _________________ file I/O _________________

What is Professor Ord’s favorite beer? (1 pt)

When the following program is run on a Sun SPARC Unix system and sorted by the address printed with the %p format specifier, specify the order of the lines printed from smallest value to largest value. (2 points each)

```c
void foo( int, int ); /* Function Prototype */
static int a;

int main( int argc, char *argv[] ) {
    int b = 7;
    static int c = 11;
    foo( argc, c );

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

void foo( int d, int e ) {
    int f = 5;
    int g;

    /* 6 */ (void) printf( "6: g --> %p\n", &g );
    /* 7 */ (void) printf( "7: a --> %p\n", &a );
    /* 8 */ (void) printf( "8: e --> %p\n", &e );
    /* 9 */ (void) printf( "9: d --> %p\n", &d );
    /* 10 */ (void) printf( "10: f --> %p\n", &f );
```
Extra Credit
Optimize the following assembly code fragment. (up to 3 pts)

```
mov %l2, %o0
mov 256, %o1
call .mul
nop
mov %o0, %l2
```

Optimized version here

What is the value of each of the following expressions? (1 pt each)

```
char *a = "End this, please!";           /*  char a[] = "End this, please!";  */
printf( "%c\n", *"Damn this is weird!" - 2 );
printf( "%c\n", toupper( "The Big Push!"[9] ) );
printf( "%c\n", *("The Big Push!" + 8) );
printf( "%c\n", 5["I loved CSE 30!" ] );
printf( "%c\n", *(a + 5) );
printf( "%c\n", toupper( 4[a] ) );
```

Tell me something you learned in this class that is extremely valuable to you and that you think you will be able to use for the rest of your programming/computer science career. (1 point if serious; you can add non-serious comments also)
<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