CSE 30
Winter 2004
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 and 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 ___________________ (12 points)
Total ___________________
1. Number Systems

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

binary ______________________________________
octal ______________________________________
decimal ______________________________________

Convert $-478_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

sign-magnitude ______________________________________
1’s complement ______________________________________
2’s complement ______________________________________

Convert $391_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 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)

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

<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>
</tbody>
</table>

3. Branching

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

```
SPARC Assembly

! map x to %l1
! map y to %l5

mov -25, %l1
mov -15, %l5
cmp %l5, -5
bg endloop
nop

loop:
cmp %l1, %l5
bg endif
nop
mov %l5, %o0
call bar
nop
mov %o0, %l1

endif:
add %l5, 6, %l5
cmp %l5, -5
ble loop
nop

endloop:
mov %l1, %o0
mov 5, %o1
call .div
nop
mov %o0, %l5
```

```c
int x;
int y;
```
4. Bit Operations

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

set 0xFA420DED, %l0
set 0xA8675309, %l1
or %l0, %l1, %l0

Value in %l0 is _______________________________________  (2 points)

set 0xFA420DED, %l0
srl %l0, 13, %l0

Value in %l0 is _______________________________________  (2 points)

set 0xFA420DED, %l0
sra %l0, 9, %l0

Value in %l0 is _______________________________________  (2 points)

set 0xFA420DED, %l0
set 0x??????????, %l1
btog %l1, %l0  
! Value in %l0 is now OxDEAFCEDE

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

set 0xFA420DED, %l0
set 0xA8675309, %l1
and %l0, %l1, %l0

Value in %l0 is _______________________________________  (2 points)

set 0xFA420DED, %l0
sll %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)

```asm
.global main
/* main.s */
.section ".text"

main:
    save %sp, -96, %sp
    mov 3, %o0
    call recurse
    nop

    ret

restore

.global recurse
/* recurse.s */

.section ".rodata"
dash: .asciz "-"
percent_d: .asciz "%d"
NL: .asciz "\n"

.section ".text"
recurse:
    save %sp, -96, %sp
    cmp %i0, %g0
    ble end
    nop
    sub %i0, 1, %o0
    call recurse
    nop
    set percent_d, %o0
    mov %i0, %o1
    call printf
    nop

    mov %g0, %l0
    cmp %l0, %i0
    bge endloop
    nop
    loop:
    set dash, %o0
    call printf
    nop
    inc %l0
    cmp %l0, %i0
    bl loop
    nop
    endloop:
    set NL, %o0
    call printf
    nop

    sub %i0, 1, %o0
    call recurse
    nop

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

local_stack_var2 = &local_stack_var3.s1[1]; /* 1 */
local_stack_var1 = local_stack_var3.s3 + y;    /* 2 */
local_stack_var3.s2 = x + local_stack_var3.s4; /* 3 */
return ( y + *local_stack_var2++ );            /* 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 if an unsigned value is a power of 2 or not. 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 = 1;

    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 \(-151.875\) (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 0x43436000 (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).

- **xorcc %g5, -13, %o2**: __________________________ (5 points)
- **sth %o2, [%l4 + %i3]**: __________________________ (5 points)

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

- **0xF452C016**: __________________________ (5 points)
- **0x24BFFFF6**: __________________________ (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
int a;
int b = 911;
static int c;
int d;
static int foo( int e ) {
    static int f;
    static double g = 4.20;
    int (*h)(int) = foo;
    int *i;
    i = (int *) malloc( b );
    ...
}
```

Fill in the letter corresponding to the correct **scoping/visibility** for each of the variables:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) Global across all modules/functions linked with this source file.</td>
<td>B) Global just to this source file.</td>
<td>C) Local to function foo().</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fill in the letter corresponding to the correct **lifetime** for each of the variables:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
<th>foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) Exists from the time the program is loaded to the point when the program terminates.</td>
<td>B) Exists from the time function foo() is called to the point when foo() returns.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
11. Load/Store/Memory
What gets printed in the following program? (9 points)

```
.global main

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

c:    .byte 0x44

.align 2
s:    .half 0xCEDE

.align 4
i1:   .word 0xABCD99EF
i2:   .word 0xABCD99EF
i3:   .word 0xABCD99EF
x:    .word 0x00007777

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

set   i1, %l0
set   s, %l1
ldsh  [%l1], %l1
sth   %l1, [%l0]
stb   %l1, [%l0+3]

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

set   i2, %l0
set   c, %l1
ldub  [%l1], %l1
stb   %l1, [%l0+2]
stb   %l1, [%l0]

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

set   x, %l0
set   i3, %l1
ldsb  [%l1+1], %l2
sth   %l2, [%l0]
stb   %l2, [%l0+3]

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

ret
restore
```
12. Miscellaneous

The _____________ translates ______________ to ______________ addresses and caches them in the TLB. (6 points)

A _________-Endian machine stores the MSB (most significant byte) of a word at a lower memory location than the LSB (least significant byte). (2 points)

What gets printed?

```c
int x = 7;
printf( "%c", '0' + x );
```

(2 points)

A _____________________________ cache ensures main memory is the same as the cache copy. (2 points)

At the end of the last lecture, the class chose a not-so-secret bonus point password. What is it? (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 = 911;

int main( int argc, char *argv[] ) {

    int b = 911;
    int c = 405;

    foo( c, argc );

    /* 1 */ (void) printf( "a --> %p\n", &a );
    /* 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 );
    /* 6 */ (void) printf( "c --> %p\n", &c );
}

void foo( int d, int e ) {

    int f = a;
    static int g;

    /* 7 */ (void) printf( "g --> %p\n", &g );
    /* 8 */ (void) printf( "e --> %p\n", &e );
    /* 9 */ (void) printf( "f --> %p\n", &f );
    /* 10 */ (void) printf( "d --> %p\n", &d );
}
```

prints

smallest value

largest value
Extra Credit
What does the following SPARC assembly language program output?

```assembly
.global main

.section ".rodata"

code: .asciz "1100101001000010"
result: .asciz "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
fmt: .asciz "%c"

.section ".text"
main:
save %sp, -96, %sp
set code, %l0
top:
  ldub [%l0], %l1
clr %l2
clr %l3
set result, %l4
  cmp %l1, %g0
  be end
  nop
  loop:
  mov %l2, %o0
  mov 2, %o1
  call .mul
  nop
  sub %l1, 0x30, %l1
  add %o0, %l1, %l2
  inc %l3
  inc %l0
  ldub [%l0], %l1
  cmp %l3, 4
  bl loop
  nop
  set fmt, %o0
  ldub [%l4+%l2], %o1
  call printf
  nop
  ba top
  nop
end:
  ret
restore
```

Output ________________________ (4 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)
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>7D ]</td>
<td>7E ~</td>
<td>7F DEL</td>
<td></td>
</tr>
</tbody>
</table>
Scratch Paper