CSE 30
Spring 2003
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 Subroutines and Calling Convention Hacking ___________________ (15 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 ___________________ (32 points)

SubTotal ___________________ (216 points)
Extra Credit ___________________ (10 points)
Total ___________________
1. Number Systems

Convert $\text{FADE}_{16}$ (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.** (6 points)

- Sign-magnitude: ____________________________
- 1’s complement: ___________________________
- 2’s complement: ___________________________

Convert $272_{10}$ 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)

\[
\begin{array}{c}
01111011 \\
+10111001 \\
\hline
-----------
\end{array}
\quad
\begin{array}{c}
11010110 \\
+10010100 \\
\hline
-----------
\end{array}
\quad
\begin{array}{c}
01010101 \\
+00101011 \\
\hline
-----------
\end{array}
\]

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

3. Branching

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

\[
\begin{center}
\text{C} \\
\begin{verbatim}
int a; int b;
for ( a = 15; a <= 75; a = a + 2) {
    b = foo( a );
    if ( b >= a )
        a = a / 5;
}
\end{verbatim}
\end{center}
\]
4. Bit Operations

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

```
set  0x420FADED, %l0
set  0xFADED420, %l1
or  %l0, %l1, %l0
Value in %l0 is _______________________________________  (2 points)
```

```
set  0x420FADED, %l0
sll  %l0, 14, %l0
Value in %l0 is _______________________________________  (2 points)
```

```
set  0x420FADED, %l0
srl  %l0, 7, %l0
Value in %l0 is _______________________________________  (2 points)
```

```
set  0x420FADED, %l0
set  0x????????, %l1
btog %l1, %l0   ! Value in %l0 is now 0xFEEDBABA
Value set in %l1 must be this bit pattern _____________________________  (3 points)
```

```
set  0x420FADED, %l0
set  0xFADED420, %l1
and  %l0, %l1, %l0
Value in %l0 is _______________________________________  (2 points)
```

```
set  0xFADED420, %l0
sra  %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)

```
.global main  /* main.s */

.main:
  save  %sp, -92 & -8, %sp
  set  41854276, %o0
  call  recurse
  nop
  ret
  restore

.global recurse  /* recurse.s */

.recurse:
  save  %sp, -(92 + 8) & -8, %sp

.skip:
  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
  mov  %o0, %i0
  ld  [%fp - 4], %l0
  andcc  %l0, 0x01, %g0
  bne  skip
  nop
  cmp  %i0, %g0
  be  no_go
  nop
  mov  %i0, %o0
  call  recurse
  nop
  st  %o0, [%fp - 8]

.no_go:
  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], %i0
  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
int fubar( long x, char y ) {
    short  *local_stack_var1;
    long    local_stack_var2;
    struct foo {
        char   s1;
        short  s2[3];
        short  s3;
        long   s4;
    }       local_stack_var3;

    y = local_stack_var3.s1;          /* 1 */
    local_stack_var1 = &local_stack_var3.s3;  /* 2 */
    local_stack_var2 = local_stack_var3.s2[2] + 17;  /* 3 */
    return ( local_stack_var3.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 Hacking

In PA3, you wrote a recursive printEnglish() in assembly. In a generic printEnglish(), you would typically check the sign of the parameter value. If it is negative, output the string "minus " and change the value to a positive number to be used throughout. As you recurse (call printEnglish) to the same printEnglish: entry point, you would be performing the same check over and over again, although you only need to perform this check the first time printEnglish() is called. Modify the following general template for recursive printEnglish.s to avoid the repetitive checks but still keep the recursive nature of the algorithm. You do not want to use a global/static flag to indicate whether you have already made this check [this would be bad form anyway]. You not want to add a helper function [you are an assembly hacker after all]. You can only add instructions/labels in the boxed area and change the label of the recursive call. (15 points)

```assembly
/*
 * printEnglish.s                               void printEnglish( long num );
 */
.globl printEnglish

.sect ".rodata"
/* Define all the english words/strings, etc. here. */

.sect ".text"
printEnglish:
    save  %sp, -(92 + 4) & -8, %sp
    tst   %i0       /* Testing if negative */
    bpos  positive  /* If positive, skip over the minus/negative code */
    nop
    set   minus, %o0 /* Print out the string "minus " */
    call  printf
    nop
    neg   %i0       /* Change the negative value to positive */
    /* This space blank in original version */

positive:
    /* Grab and save right-most digit with .rem here. */
    /* Get rid of right-most digit with .div here. */
    cmp   %o0, 0     /* No more digits left */
    be    no_recursion
    nop
    call  ____________________ /* recurse; call printEnglish in original version */
    nop

no_recursion:
    /* Print out the english word equivalent for the current stack frame’s saved right digit */
    ret
    restore
```
8. Floating Point

Convert 119.375\textsubscript{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 0xC376C000 (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).

xor %g3, -12, %o5 ________________________________ (5 points)

std %o2, [%l4 + %i3] ________________________________ (5 points)

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

0xF62A4016 ________________________________ (5 points)

0x9A868012 ________________________________ (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 = 411;  
static int b = 404;  
static int c;  
int d;  
static int foo( int e ) {  
    static double f = 4.20;  
    static int g;  
    int *h;  
    h = (int *) malloc( b );  
    int (*i)(int) = foo;  
    ...  
}
```

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 0xXXXXXXXX

c: .byte 0x33
.align 2
s: .half 0x9876
.align 4
i1: .word 0xabcdef44
i2: .word 0xabcdef44
i3: .word 0xabcdef44
x: .word 0x5555

.section ".text"
main:
save %sp, -96, %sp
set i1, %l0
set s, %l1
ldsh [%l1], %l1
st %l1, [%10]
stb %l1, [%10+1]
set fmt, %o0
ld [%10], %o1
call printf ________________________
nop

set i2, %l0
set c, %l1
ldsb [%l1], %l1
sth %l1, [%10+2]
stb %l1, [%10+1]
set fmt, %o0
ld [%10], %o1
call printf ________________________
nop

set x, %l0
set i3, %l1
ldsb [%l1+1], %l2
sth %l2, [%10]
stb %l2, [%10+3]
set fmt, %o0
ld [%10], %o1
call printf ________________________
nop

ret
restore
```
12. Miscellaneous
Which type of Control Unit is most likely found in RISC architectures? ______________________________

Put the following in the correct order/sequence using the numbers to the left of each word:

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

_____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____

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

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

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

void foo( int d, int e ) {
    int f = 911;
    int g;
    /*  7 */ (void) printf( "e --> %p
", &e );
    /*  8 */ (void) printf( "d --> %p
", &d );
    /*  9 */ (void) printf( "g --> %p
", &g );
    /* 10 */ (void) printf( "f --> %p
", &f );
}
```
Extra Credit

What does the following SPARC assembly language program output?

```
.global main

.section ".rodata"
fmt: .asciz "%c"
.align 4
foo: .word 0x43005364, 0x4565336B, 0x3061204E, 0x52207566, 0x6C726575, 0x73530000

.section ".text"

main: main:
    save %sp, -96, %sp
    set foo, %l0
    mov 0, %l1
    clr %l2
    mov 21, %l3
    sub %l3, %l2, %l4
    ba test
    nop

loop:
    set fmt, %o0
    ldub [%l0+%l4], %o1
    call printf, 2
    nop
    inc %l1
    mov %l1, %o0
    mov 2, %o1
    call .mul
    nop
    mov %o0, %l2
    sub %l3, %l2, %l4
    test:
    ldub [%l0+%l4], %o1
    tst %o1
    bne loop
    nop
    set fmt, %o0
    mov 0x0A, %o1
    call printf, 2
    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)
<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