CSE 30
Winter 2002
Final Exam

1. Number Systems ___________________  (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching ___________________  (18 points)
4. Bit Operations ___________________  (10 points)
5. Recursion/SPARC Assembly ___________________  (10 points)
6. Local Variables, The Stack and Return Values ___________________  (20 points)
7. SPARC Subroutines and Calling Convention ___________________  (14 points)
8. Floating Point ___________________  (18 points)
9. Linkage, Scope, Lifetime, Data ___________________  (31 points)
10. Machine Instructions ___________________  (20 points)
11. I/O & Virtual Memory ___________________  (11 points)
12. Miscellaneous ___________________  (26 points)

SubTotal ___________________  (205 points)
Extra Credit ___________________  (10 points)
Total ___________________
1. Number Systems

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

binary_______________________________________

octal ______________________________________________________________________

decimal ______________________________________________________________________

Convert $+747_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

sign-magnitude_________________________________________________________________

1’s complement________________________________________________________________

2’s complement________________________________________________________________

Convert $-456_{10}$ 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)

\[
\begin{array}{ccc}
10111001 & +01010110 & \text{---------} \\
+01010111 & +01010111 & \text{---------} \\
01000101 & +10111011 & \text{---------} \\
\end{array}
\]

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

3. Branching

Write the C statements to perform the following SPARC assembly statements. Do not optimize. (18 points)

```
  mov  75, %l2

L1:
  StatementX
  cmp  %l2, 19
  ble  L2
  nop

  StatementY
  ba   L3
  nop

L2:
  inc  %l2
  StatementZ

L3:
  sub  %l2, 14, %l2

L4:
  cmp  %l2, %g0
  bge  L1
  nop

/* %l2 mapped to local variable x */
```
4. Bit Operations

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

set 0xDB7E9F43, %l0
set 0x78CDCD78, %l1
and %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xDB7E9F43, %l0
sll %l0, 7, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xDB7E9F43, %l0
sra %l0, 13, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xDB7E9F43, %l0
set 0x91817161, %l1
bset %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xDB7E9F43, %l0
set 0x928276B5, %l1
xor %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)
5. Recursion/SPARC Assembly

Given main.s and fubar.s, what gets printed when executed? (10 points)

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

.section ".rodata"
.align 4

code: .word 0x43216153, 0x21764567, 0x61336E4A, 0x306F2020, 0x4C3E5220, 0x206F6543
.word 0x6376006B, 0x6900734C, 0x00000030

.section ".text"
main:
  save  %sp, -92 & -8, %sp
  set   code, %o0
  mov   31, %o1
  call  fubar
  nop

  ret
  restore

/* fubar.s */

.global fubar

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

.section ".text"
fubar:
  save  %sp, -(92 + 1) & -8, %sp
  dec   %i1
  cmp   %i0, %g0
  be    end
  nop
  cmp   %i1, %g0
  bl    end
  nop
  ldub  [%.i0 + %.i1], %.l0
  stb   %.l0, [%.fp - 1]
  add   %.i1, -2, %.o1
  mov   %.i0, %.o0
  call  fubar
  nop
  set   fmt, %.o0
  ldub  [%.fp - 1], %.o1
  call  printf
  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
int fubar( int a, int b ) {
    int *local_stack_var1;
    unsigned short local_stack_var2;
    struct foo {
        int s1;
        short s2[7];
        char s3;
        long s4;
    } local_stack_var3;

    local_stack_var3.s3 = 'B'; /* Use the ASCII value, not 'B' */
    local_stack_var2 = local_stack_var3.s2[3];
    local_stack_var3.s4 = *local_stack_var1++;
    return ( local_stack_var3.s1 - b );
}
```

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. (20 points)
7. SPARC Subroutines and Calling Convention (2 point each)

In most architectures, some of the Stack Frame is built by the calling function and some of the Stack Frame is built by the called function. The same is generally true of the SPARC architecture. Fill in the blanks for the general model of Stack Frame creation/function calling convention.

The ______________ function allocates space on the Stack for local variables and performs any initialization.

The ______________ function calls the instruction to set the program counter with the saved return address.

The ______________ function deallocates the space on the Stack used for the arguments

The ______________ function allocates space on the Stack for the arguments being passed in the function call.

The ______________ function allocates space for and possibly saves values of registers that it may use.

The ______________ function places the return value in the architecture-specific return value area.

The ______________ function calls the instruction to save the current value of the program counter to be used as the return address.

8. Floating Point

Convert -109.875 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 0x43D36000 (single-precision IEEE floating-point representation) to fixed-point decimal.

fixed-point decimal ____________________________________ (6 points)

Complete the following initializations of the parts of a single-precision IEEE floating-point representation: 

NOTE: Use only shift operations! (2 pts each)

```c
void classifyFloatParts( float f ) {
    unsigned int bits = *(unsigned int *) &f; /* bits is the bit pattern of f */
    unsigned int sign = ________________________________; /* sign bit */
    unsigned int exp  = ________________________________; /* exponent bits */
    unsigned int frac = ________________________________; /* mantissa/fraction bits */
    ...
}
```
9. Linkage, Scope, Lifetime, Data

For the following program fragment, specify what C runtime area/segment will be used for each variable definition or statement: (31 points — 1 point each)

```c
static int a;
int b = 4;
static int c = 20;
int d;

int foo( int e ) {
    static double f = 4.20;
    int g = 420;
    int *h;
    int i;
    h = (int *) malloc( g );
    ...
}
```

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 ______
```
10. 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).

add %i3, %i5, %o1 ___________________________________  (5 points)

ld [%o1], %i4 ___________________________________  (5 points)

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

0x96A6FFF6 ___________________________________  (5 points)

0x36BFFFFB ___________________________________  (5 points)

11. I/O & Virtual Memory

Which combination of two of the following is fastest/most efficient for large data transfers.

Programmed I/O
Interrupt-driven _________________________ / _________________________ (2 points)
DMA
CPU blocked (Polling)

Paging and swapping refer to parts of or entire program images being moved back and forth between what 2 storage areas/hierarchies? (2 points)

_________________________  and  _________________________

A ________ _________ occurs because a page is needed but was not found to be in main memory. (1 points)

The ____________ translates ____________ to ____________ addresses. (3 points)

The ____________ caches these recently translated addresses. (1 points)

This type of I/O uses regular instructions to perform I/O and not any special instructions. (1 points)

___________________________________

A ____________________ cache ensures the data in the cache is the same as what is in main memory. (1 points)
12. Miscellaneous

Circle T for true; Circle F for false; +2 if correct, 0 if incorrect or blank, -1 if both T and F are circled!

T or F One way to generally speed up string compares on a Little-Endian machine is to use word compares instead of byte by byte compares. (2 points)

T or F The hidden bit in IEEE floating-point representation allows us to more precise values. (2 points)

T or F Static linked binaries are usually larger than dynamic linked binaries. (2 points)

T or F Doubling the clock rate on a specific CPU will mean all instructions will execute twice as fast. (2 points)

T or F The resulting Text segment of a program compiled on a CISC system is generally smaller than that of the same program compiled on a RISC system. (2 points)

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 values. (2 points each)

```c
void foo( int fubar );

int a = 420;

int main( int argc, char *argv[] ) {
    int x = 911;
    static int b;

    foo( x );

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

void foo( int fubar ) {
    int y;

    /* 7 */ (void) printf( "y --> %p\n", &y );
    /* 8 */ (void) printf( "fubar --> %p\n", &fubar );
}
```
Extra Credit

What does the following SPARC assembly language program output?

```assembly
.global main

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

.align 4
foo:   .word 0x52006941, 0x706C6C69, 0x6565796E, 0x00737500, 0x6C766569, 0x734C0000

.section ".text"
main:
    save %sp, -92 & -8, %sp
    mov 1, %l1
    clr %l2
    set foo, %l0
    ba test
    nop

loop:
    set fmt, %o0
    ldub [%l0+%l2], %o1
    call printf, 2
    nop
    mov %l1, %o0
    mov 2, %o1
    call .mul
    nop
    mov %o0, %l2
    inc %l1

test:
    ldub [%l0+%l2], %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>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>
Scratch Paper
Scratch Paper