CSE 30  
Fall 2007  
Final Exam

1. Number Systems
   ___________________ (25 points)

2. Binary Addition/Condition Code Bits/Overflow Detection
   ___________________ (12 points)

3. Branching
   ___________________ (19 points)

4. Bit Operations
   ___________________ (13 points)

5. Recursion/SPARC Assembly
   ___________________ (10 points)

6. Local Variables, The Stack, Return Values
   ___________________ (20 points)

7. More Recursive Subroutines
   ___________________ (12 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
    ___________________ (29 points)

SubTotal
   ___________________ (213 points)

Extra Credit
   ___________________ (10 points)

Total
   ___________________
1. Number Systems

Convert $\text{FB4}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

- **binary**
- **octal**
- **decimal** (convert to signed decimal)

Convert $432_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

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

Convert $-389_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

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

C Compiling Sequence

Put the following in the correct order/sequence using the numbers to the left of each word: (10 points)

A. executable (.exe/a.out)  B. assembler  C. resulting .o file
D. loader  E. program execution  F. resulting .s file
G. compiler  H. preprocessor  I. source code (.c file)  J. linkage editor

_____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____
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)

\[
\begin{array}{ccc}
00101101 & +11010100 & \text{---------} \\
00110111 & +01001001 & \text{---------} \\
11101001 & +10010111 & \text{---------} \\
\end{array}
\]

\[
\begin{array}{ccccc}
| \text{N} | \text{Z} | \text{V} | \text{C} | \\
\hline
|   |   |   |   | \\
\hline
|   |   |   |   | \\
\hline
|   |   |   |   | \\
\end{array}
\]

3. Branching

Given the following C code write the equivalent function in unoptimized SPARC Assembly. Just perform a direct translation using the proper if-else structure discussed in class/notes (think opposite logic). (19 points)

C

```c
int checkIfEvenAndPositive( int value )
{
    if ( ((value % 2) == 0) && (value > 0) )
        return 1;
    else
        return 0;
}
```

SPARC Assembly

```assembly
.int
.global checkIfEvenAndPositive
.section ".text"
checkIfEvenAndPositive:
    save %sp, -96, %sp
```

In C, how could you rewrite/replace the expression

\[(value \% 2) == 0\]

to check if value is even such that the new expression is more optimal and does not translate into a function call or an expensive operation in SPARC Assembly? Give the more optimal expression:
4. Bit Operations

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

```
set 0xCAFEBABE, %l0
set 0x87654321, %l1
or  %l0, %l1, %l0

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

```
set 0xCAFEBABE, %l0
srl %l0, 13, %l0

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

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

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

```
set 0xCAFEBABE, %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 0xCAFEBABE, %l0
set 0x87654321, %l1
and %l0, %l1, %l0

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

```
set 0xCAFEBABE, %l0
sra %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
.globl main /* main.s */
.section "".text"
main:
    save %sp, -96, %sp
    mov 19, %o0
    call recurse
    mov
    ret
    restore

.globl recurse /* recurse.s */
.section "".rodata"
fmt: .asciz "%d"
.section "".text"
recurse:
    save %sp, -96, %sp
    cmp %i0, 2
    bge L1
    mov
    ret
    restore

L1:
    sra %i0, 1, %o0
    call recurse
    mov %i0, %o1
    set fmt, %o0
    call printf
    nop
    ret
    restore
```

What gets printed? _____________________
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;
    int    s2;
    char   s3[3];
    short  s4[3];
  }       local_stack_var3;
  local_stack_var2    = local_stack_var3.s4 + y; /* 1 */
  local_stack_var3.s1 = x + *++local_stack_var2; /* 2 */
  local_stack_var1    = &local_stack_var3.s3[1]; /* 3 */
  return ( local_stack_var3.s2 + 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. (20 points)
What is the output of the following program? (12 pts)

```c
#include <stdio.h>

int
AAA( int a )
{
    int result1;
    printf( "a = %d\n", a );
    if ( a <= 1 )
        return 1;
    else {
        result1 = (BBB( a - 1 ) + a);
        printf( "result1 = %d\n", result1 );
        return result1;
    }
}

int
BBB( int b )
{
    int result2;
    printf( "b = %d\n", b );
    if ( b <= 1 )
        return 3;
    else {
        result2 = (AAA( b - 3 ) + b);
        printf( "result2 = %d\n", result2 );
        return result2;
    }
}

int
main( int argc, char *argv[] )
{
    printf( "%d\n", AAA( 10 ) );
    return 0;
}
```

Put output here
8. Floating Point

Convert $-119.125_{10}$ (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 0x434DC000 (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).

sra    %l2, 7, %o3 ______________________________________   (5 points)

ldsb   [%fp - 15], %l1 ____________________________________   (5 points)

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

0xF43A8013 ___________________________________________   (5 points)

0x3CBFFPFC ___________________________________________   (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
static int a = -9; ____________
int b; ____________
int c = a; ____________
static int d; ____________
static int foo( int e ) { ____________ ( foo ) ____________ ( e )
static double f = 4.20; ____________
static int g;
int *h;
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().

```
foo ______
i ______
h ______
g ______
f ______
e ______
d ______
c ______
b ______
a ______
```

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.

```
foo ______
i ______
h ______
g ______
f ______
e ______
d ______
c ______
b ______
a ______
```
11. Load/Store/Memory
What gets printed in the following program? (9 points)

.global main

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

.c:     .byte   0xA9

.s:     .half   0xFEDA

.i1:    .word   0x915ACDC9
.i2:    .word   0x915ACDC9
.i3:    .word   0x915ACDC9
.x:     .word   0x00005678

.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
ldub  [%l1], %l2
stb   %l2, [%10]
sth   %l2, [%10+2]

.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
What is the output of the following program? (8 points)

```c
#include <stdio.h>
#define SIZE 4
void mystery( unsigned char bytes[], size_t size );

int main()
{
 int i;
 unsigned char bytes[SIZE] = { 0x19, 0x42, 0x08, 0x37 };

 mystery( bytes, SIZE );

 for ( i = 0; i < SIZE; ++i )
 { 
  printf( "0x%X\n", bytes[i] );
  putchar('\n');
  return 0;
 }

 void mystery( unsigned char bytes[], size_t size )
 {
  int i;

   for ( i = 0; i < size; ++i )
    bytes[i] = (bytes[i] >> 4) | ((bytes[i] & 0xF) << 4) ;
 }
```

Who is Rick's favorite rap artist? (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
int main( int argc, char *argv[] ) {

    int b = 69;
    int c;

    foo( c, b );

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

void foo( int d, int e ) {

    int f = e;
    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 );
}
```
Extra Credit (10 points)
Optimize the following SPARC assembly program starting after the save instruction. (4 points)

```assembly
.global main
.section ".rodata"
fmt: .asciz "%c"
.align 2
foo: .half 0x6465, 0x6B61, 0x4E20,
     0x6672, 0x7553, 0x0000
.section ".text"
main:
    save   %sp, -96, %sp
    clr    %l1
    set    foo, %l0
    ldub   [%l0+%l1], %o1
    tst    %o1
    be     end
    nop
    loop:
    set    fmt, %o0
    call   printf, 2
    nop
    inc    %l1
    ldub   [%l0+%l1], %o1
    tst    %o1
    bne    loop
    nop
    end:
    set    fmt, %o0
    mov    0x0A, %o1
    call   printf, 2
    nop
    mov    0, %i0
    ret
    restore
```

What is the value of each of the following expressions taken sequentially based on changes that may have been made in previous statements? (6 points)

```c
char *a = "BuildOnACommit!";
char *p = a + 4;

printf( "%c\n", *p);  ______
printf( "%c\n", *++p );  ______
printf( "%c\n", *&(2[p++]) );  ______
printf( "%c\n", *(a+3) );  ______
printf( "%c\n", *(p+1) );  ______
printf( "%c\n", *a++ );  ______
```
<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