By filling in the above and signing my name, I confirm I will complete this exam with the utmost integrity and in accordance with the Policy on Integrity of Scholarship.

CSE 30
Spring 2012
Final Exam

1. Number Systems / C Compiling Sequence  (32 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 (11 points)
12. Miscellaneous (29 points)

SubTotal (222 points)
Extra Credit (5+%) (12 points)
Total

This exam is to be taken by yourself with closed books, closed notes, no electronic devices. You are allowed both sides of an 8.5"x11" sheet of paper handwritten by you.
1. Number Systems

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

- **binary**_____________________________________ (straight base conversion)
- **octal**_______________________________________ (straight base conversion)
- **decimal**____________________________________ (convert to signed decimal)

Convert $488_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

- **sign-magnitude**

- **1’s complement**

- **2’s complement**

Convert $-327_{10}$ 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 letters to the left of each word: (11 points)

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

____ —> ____ —> ____ —> ____ —> ____ —> ____ —> ____ —> ____ —> ____ —> ____ —> ____

Rt-Lt Rule

Using the C Rt-Lt Rule, define a variable named foo that is a pointer to a function that takes a single argument of type pointer to char and returns a pointer to an array of 5 elements where each element is of type pointer to struct fubar. (6 points)
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)

\[
\begin{array}{cccccc}
10110100 & +00101101 & \text{---------} \\
10110100 & +10110100 & \text{---------} \\
10110100 & +11001100 & \text{---------} \\
\end{array}
\]

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

3. Branching

Given the following SPARC Assembly code write the equivalent function in C. Just perform a direct translation using the proper if-else structure discussed in class/notes. Assume all parameters and return types are int. Name the parameters \( a \) and \( b \). (19 pts)

```
.global foo
.section ".text"
foo:
save  %sp, -96, %sp

cmp   %i0, %i1
ble   else

nop

mov   %i0, %o0
mov   %i0, %o1
call  .mul

nop

mov   %o0, %i1
sub   %i1, 4, %i0
ba    end

else:

add   %i0, 9, %i1

mov   %i1, %o0
mov   17, %o1
call  .div

nop

mov   %o0, %i0

end:

add   %i0, %i1, %i0
ret
restore
```
4. Bit Operations

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

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

Value in %l0 is _______________________________________ (2 points)

```
set 0xFACEBAAD, %l0
sra %l0, 9, %l0
```

Value in %l0 is _______________________________________ (2 points)

```
set 0xFACEBAAD, %l0
sll %l0, 14, %l0
```

Value in %l0 is _______________________________________ (2 points)

```
set 0xFACEBAAD, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0 ! Value in %l0 is now OxCAFEBABE
```

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

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

Value in %l0 is _______________________________________ (2 points)

```
set 0xFACEBAAD, %l0
srl %l0, 10, %l0
```

Value in %l0 is _______________________________________ (2 points)
5. Recursion/SPARC Assembly
Given `main.s` and `fubar.s`, what gets printed when executed? (10 points)

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

.section ".rodata"
.align 4
code: .word 0x65766950, 0x65676146, 0x00336E4A, 0x306F2020, 0x4C3E5220, 0x20756543
       .word 0x6C760065, 0x6900734C, 0x00000030

.main:
    save %sp, -92 & -8, %sp
    set code, %o0
    mov 0, %o1
    call fubar
    nop
    ret
    restore

.global fubar
     /* fubar.s */

.fmt: .asciz "%c"

.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 + %i10 ], %l1
    stb %l1, [ %fp - 1 ]
    cmp %l0, 0x04
    be end
    nop
    add %i1, 1, %o1
    mov %i0, %o0
    call fubar
    nop

end:
    set fmt, %o0
    ldub [ %fp - 1 ], %o1
    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
long fubar( char *x, int y ) {
    int  *local_stack_var1;
    struct foo {
        short s1[5];
        char  s2;
        char  *s3;
        int   s4;
    }     local_stack_var2;
    int   local_stack_var3;

    local_stack_var2.s2 = *x + 5; /* 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 */
}
```

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)

#include <stdio.h>

int
AAA( int a )
{
    int result1;
    printf( "a = %d\n", a );
    if ( a >= 10 )
        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 >= 10 )
        return 3;
    else {
        result2 = (AAA( b + 3 ) + b);
        printf( "result2 = %d\n", result2 );
        return result2;
    }
}

int
main( int argc, char *argv[] )
{
    printf( "%d\n", BBB( 1 ) );
    return 0;
}
8. Floating Point

Convert 146.75\text{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 0xC342E000 (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).

\text{andcc} %o4, 0x17, %i4 __________________________________   (5 points)

\text{std} %l6, [%fp + %o1] __________________________________   (5 points)

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

0x3c800011 __________________________________   (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 = 99; ______________
int b; ______________
static int c = 37; ______________
static int d; ______________
static int foo( int e ) {   ____________
   static double f; ______________
   static int g = 17; ______________
   int *h; ______________
   h = (int *) malloc( e ); ______________
   (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 _______ a _______
b _______
c _______
d _______
e _______
f _______
g _______
h _______
i _______

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 _______ a _______
b _______
c _______
d _______
e _______
f _______
g _______
h _______
i _______
11. Load/Store/Memory
Specify the hex values requested after those lines have been fully executed. (11 points)

.global main

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

c: .byte 0xBB
      .align 2
s: .half 0xCAFE
      .align 4
i1: .word 0x12345678
i2: .word 0x12345678
i3: .word 0x12345678
x: .word 0x77770000

.section ".text"
main:
  save %sp, -96, %sp
  set  x, %l0
  set  s, %l1
  ldsh [%l1], %l2  _____________________ Hex value in %l2
  stb  %l2, [%l0+1]  _____________________  Hex value in word labeled x
  srl  %l2, 12, %l2  _____________________  Hex value in %l2
  stb  %l2, [%l0+3]
  set  fmt, %o0
  ld  [%l0], %o1
  call printf  _____________________ Hex value in word labeled x
  nop  (same as output of this printf)
  set  i1, %l0
set  c, %l1
ldub [%l1], %l2  _____________________ Hex value in %l2
  sth  %l2, [%l0+2]
  stb  %l2, [%l0]
  set  fmt, %o0
  ld  [%l0], %o1
  call printf  _____________________ Hex value in word labeled i1
  nop  (same as output of this printf)
  set  i2, %l0
set  i3, %l1
ld  [%l1], %l2  _____________________ Hex value in %l2
  sth  %l2, [%l0]
  sra  %l2, 16, %l2  _____________________  Hex value in %l2
  stb  %l2, [%l0+1]
  set  fmt, %o0
  ld  [%l0], %o1
  call printf  _____________________ Hex value in word labeled i2
  nop  (same as output of this printf)

ret
 restore
12. Miscellaneous
What is the output of the following program? (8 points)

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

void mystery( unsigned short hwords[], size_t size );

int
main()
{
    int i;
    unsigned short hwords[SIZE] = { 0x1942, 0xF837, 0x13AC, 0xB6D5 };
    mystery( hwords, SIZE );
    for ( i = 0; i < SIZE; ++i )
    {
        printf( "0x%04X ", hwords[i] ); /* Prints 0xXXXX */
    }
    putchar( '\n' );
    return 0;
}

void
mystery( unsigned short hwords[], size_t size )
{
    int i;
    for ( i = 0; i < size/2; ++i )
    {
        hwords[i] = hwords[i] ^ hwords[i + 2];
        hwords[i + 2] = hwords[i + 2] ^ hwords[i];
        hwords[i] = hwords[i] ^ hwords[i + 2];
    }
}
```

Who is Rick's celebrity look-alike? (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 = 24;

int main( int argc, char *argv[] )
{
    int b = 99;
    int c;
    foo( c, b );

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

void foo( int d, int e )
{
    int f = 42;
    static int g;

    /* 6 */ (void) printf( "g --> %p\n", &g );
    /* 7 */ (void) printf( "e --> %p\n", &e );
    /* 8 */ (void) printf( "a --> %p\n", &a );
    /* 9 */ (void) printf( "f --> %p\n", &f );
    /* 10 */ (void) printf( "d --> %p\n", &d );
}
```
Extra Credit  (12 points)
Optimize the following SPARC assembly program by filling in the delay slot **nops** with useful instructions starting **after** the save instruction. (5 points)

```assembly
.global foo
.section "./text"
foo:
    save %sp, -96, %sp

    cmp %i0, %i1
    ble  else
    nop

    mov %i0, %o0
    mov %i0, %o1
    call .mul
    nop

    mov %o0, %i1
    sub %i1, 4, %i0
    ba  end
    nop

else:
    add %i0, 9, %i1

    mov %i1, %o0
    mov 17, %o1
    call .div
    nop

    mov %o0, %i0

end:
    add %i0, %i1, %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? (7 points)

```c
char a[] = "CSE131 Rules!"
char *p = a + 2;

printf( "%c", *p++ );
printf( "%c", --*p );
printf( "%c", --p[2] );
p = p + 4;
printf( "%c", *++p = a[9] + 3 );
p++;
printf( "%c", a[10] = +++p + 7 );
printf( "%d", p - a );
printf( "%s", a );
```
Hexadecimal - Character

| 00 NUL | 01 SOH | 02 STX | 03 ETX | 04 EOT | 05 ENQ | 06 ACK | 07 BEL |
| 08 BS  | 09 HT  | 0A NL  | 0B VT  | 0C NP  | 0D CR  | 0E SO  | 0F SI  |
| 10 DLE | 11 DC1 | 12 DC2 | 13 DC3 | 14 DC4 | 15 NAK | 16 SYN | 17 ETB |
| 18 CAN | 19 EM  | 1A SUB | 1B ESC | 1C FS  | 1D GS  | 1E RS  | 1F US  |
| 20 SP  | 21 !   | 22 "   | 23 #   | 24 $   | 25 %   | 26 &   | 27 '   |
| 28 (   | 29 )   | 2A *   | 2B +   | 2C ,   | 2D -   | 2E .   | 2F /   |
| 30 0   | 31 1   | 32 2   | 33 3   | 34 4   | 35 5   | 36 6   | 37 7   |
| 38 8   | 39 9   | 3A :   | 3B ;   | 3C <   | 3D =   | 3E >   | 3F ?   |
| 40 @   | 41 A   | 42 B   | 43 C   | 44 D   | 45 E   | 46 F   | 47 G   |
| 48 H   | 49 I   | 4A J   | 4B K   | 4C L   | 4D M   | 4E N   | 4F O   |
| 50 P   | 51 Q   | 52 R   | 53 S   | 54 T   | 55 U   | 56 V   | 57 W   |
| 58 X   | 59 Y   | 5A Z   | 5B [   | 5C \   | 5D ]   | 5E ^   | 5F _   |
| 60 `   | 61 a   | 62 b   | 63 c   | 64 d   | 65 e   | 66 f   | 67 g   |
| 68 h   | 69 i   | 6A j   | 6B k   | 6C l   | 6D m   | 6E n   | 6F o   |
| 70 p   | 71 q   | 72 r   | 73 s   | 74 t   | 75 u   | 76 v   | 77 w   |
| 78 x   | 79 y   | 7A z   | 7B {   | 7C |   | 7D }   | 7E ~   | 7F DEL |

A portion of the Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ postfix increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- postfix decrement</td>
<td></td>
</tr>
<tr>
<td>[] array element</td>
<td></td>
</tr>
<tr>
<td>() function call</td>
<td></td>
</tr>
<tr>
<td>* indirection</td>
<td>R to L</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td></td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td></td>
</tr>
<tr>
<td>&amp; address-of</td>
<td></td>
</tr>
<tr>
<td>sizeof size of type/object</td>
<td></td>
</tr>
<tr>
<td>(type) type cast</td>
<td></td>
</tr>
<tr>
<td>* multiplication</td>
<td>L to R</td>
</tr>
<tr>
<td>/ division</td>
<td></td>
</tr>
<tr>
<td>% modulus</td>
<td></td>
</tr>
<tr>
<td>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>
Scratch Paper
Scratch Paper