CSE 30
Spring 2008
Final Exam

1. Number Systems / C Compiling Sequence  ______________  (26 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  ______________  (216 points)

Extra Credit  ______________  (13 points)

Total  ______________
1. Number Systems

Convert $\text{FBC}_{16}$ (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 $\text{482}_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

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

Convert $-\text{329}_{10}$ 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. C source code (.c file)
- B. linkage editor
- C. resulting .o file
- D. loader
- E. program execution
- F. resulting .s file
- G. executable (.exe/a.out)
- H. assembler
- I. segmentation fault/core dump
- J. C compiler
- K. C preprocessor

___ —> ___ —> ___ —> ___ —> ___ —> ___ —> ___ —> ___ —> ___ —> ___ —> ___
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)

<table>
<thead>
<tr>
<th>10110100</th>
<th>10110100</th>
<th>10110100</th>
</tr>
</thead>
<tbody>
<tr>
<td>+00101101</td>
<td>+10110100</td>
<td>+01001100</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
</tbody>
</table>

N Z V C
|   |   |   |   |
---------

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)

**SPARC Assembly**

```
.globl foo
.sect .text
foo:
save %sp, -96, %sp
	cmp %i0, %i1
	bl   else
	nop
	mov %i0, %o0
	mov %i0, %o1

call .mul
	nop
	mov %o0, %i1
	sub %i1, 4, %i0
	ba    end
	nop

e:else:
	add %i0, 9, %i1
	mov %i1, %o0
	mov 17, %o1

call .div
	nop
	mov %o0, %i0

de:    
	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.**

```plaintext
set 0xBEADFACE, %l0
set 0x87654321, %l1
or  %l0, %l1, %l0

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

```plaintext
set 0xBEADFACE, %l0
sra %10, 9, %l0

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

```plaintext
set 0xBEADFACE, %l0
sll %10, 14, %l0

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

```plaintext
set 0xBEADFACE, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0

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

```plaintext
set 0xBEADFACE, %l0
set 0x?????????, %l1
and %l0, %l1, %l0

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

```plaintext
set 0xBEADFACE, %l0
srl %l0, 10, %l0

Value in %l0 is ____________________________ (2 points)
```
Given main.s and fubar.s, what gets printed when executed? (10 points)

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)

```c
#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;
}
```

Put output here
8. Floating Point

Convert 146.75_{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).

andcc %o4, 0x17, %i4 ______________________________________  (5 points)
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 ) { ____________ ( foo ) ______________ ( 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 _______ 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  0xXXXXXXXX

.c:    .byte 0xBB

.s:    .half 0xCAFE

.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
              (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
              (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
              (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
do,

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];
    }
}
```

If Rick had a dog, what would it's name be? (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 );
    printf( "c --> %p\n", &c );
    printf( "foo --> %p\n", foo );
    printf( "b --> %p\n", &b );
    printf( "argc --> %p\n", &argc );
    printf( "malloc --> %p\n", malloc(50) );
}

void foo( int d, int e ) {
    int f = 42;
    static int g;
    printf( "g --> %p\n", &g );
    printf( "e --> %p\n", &e );
    printf( "a --> %p\n", &a );
    printf( "f --> %p\n", &f );
    printf( "d --> %p\n", &d );
    printf( "realloc(50) --> %p\n", realloc(50) );
    printf( "realloc(50) --> %p\n", realloc(50) );
}
```
Extra Credit  (13 points)
Optimize the following SPARC assembly program by filling in the delay slot \texttt{nops} with useful instructions starting after the \texttt{save} instruction. (6 points)

```
.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)

```
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 );  ______________________
```
<table>
<thead>
<tr>
<th>Hexadecimal - Character</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

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>* indirection</td>
<td>R to L</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td></td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td>&amp; address-of</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