CSE 30
Winter 2008
Final Exam

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

SubTotal ___________________ (215 points)

Extra Credit ___________________ (10 points)

Total ___________________
1. Number Systems

Convert $FBB_{16}$ (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 $442_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

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

Convert $-379_{10}$ 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: (10 points)

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

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td>N</td>
<td>Z</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>00000001</td>
<td>+11111111</td>
<td>10101001</td>
<td>+10010111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01110111</td>
<td>+01001001</td>
<td>+10010111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+10010111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 and short-circuiting discussed in class/notes. (19 points)

C

```c
int checkIfOddOrNegative( int value )
{
    if ( ((value % 2) != 0) || (value < 0) )
        return 1;
    else
        return 0;
}
```

SPARC Assembly

```assembly
.int .global checkIfOddOrNegative
.checkIfOddOrNegative:
.save %sp, -96, %sp
.return
```

In C, how could you rewrite/replace the expression

```
(value % 2) != 0
```

to check if value is odd 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 0xDEADBEEF, %l0
set 0x87654321, %l1
or %l0, %l1, %l0
Value in %l0 is ____________________________ (2 points)

set 0xDEADBEEF, %l0
sra %l0, 9, %l0
Value in %l0 is ____________________________ (2 points)

set 0xDEADBEEF, %l0
sll %l0, 14, %l0
Value in %l0 is ____________________________ (2 points)

set 0xDEADBEEF, %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 0xDEADBEEF, %l0
set 0x87654321, %l1
and %l0, %l1, %l0
Value in %l0 is ____________________________ (2 points)

set 0xDEADBEEF, %l0
srl %l0, 10, %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 */

.section "".rodata"
.align 4
.word 0x30003300
.code: .word 0x53736476, 0x72656E74, 0x6B335061, 0x523D4E68, 0x2B20772B, 0x66374372
.word 0x302F752F, 0x43537300, 0x00000030

.main:
    save %sp, -92 & -8, %sp
    set code, %o0
    mov 30, %o1
    call recurse
    nop
    ret
    restore

.global recurse  /* recurse.s */

.section "".rodata"

.fmt: .asciz "%c"

.recurse:
    save %sp, -96, %sp
    cmp %i0, %g0
    be end
    nop
    sub %i1, 3, %i1
    ldub [%i0 + %i1], %l0
    cmp %l0, %g0
    be end
    nop
    stb %l0, [%fp - 1]
    mov %i0, %o0
    mov %i1, %o1
    call recurse
    nop
    set fmt, %o0
    ldub [%fp - 1], %o1
    inc %o1
    call printf
    nop

.end:
    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[2];
    } 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", BBB( 10 ) );
    return 0;
}
```
8. Floating Point

Convert 123.625\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 0xC34C4000 (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).

xorcc %i2, 0x16, %l3 __________________________________________ (5 points)
sth %l2, [%fp + %i3] _________________________________________ (5 points)

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

0x9DE3BFA0 _______________________________________________ (5 points)
0x3EBFFFFD _______________________________________________ (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;
static int b = 42;
static int c;
int d = a;
static int foo( int e ) {
  double f;
  int g = 17;
  static int (*h)(int) = foo;
  static int *i;
  i = (int *) malloc( e );
  ...
}
```

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
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 0x99
.s: .half 0xFACE

.i1: .word 0x9ABCD123
.i2: .word 0x9ABCD123
.i3: .word 0x9ABCD123
.x: .word 0x55550000

.section ".text"
main:
save   %sp, -96, %sp
set    x, %l0
set    s, %l1
lduh   [%l1], %l2 _____________________ Hex value in %l2
stb    %l2, [%l0+3]  _____________________  Hex value in word labeled x
srl    %l2, 4, %l2  _____________________ Hex value in %l2
stb    %l2, [%l0+1]
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
ldsb    [%l1], %l2 _____________________ Hex value in %l2
sth    %l2, [%l0]  _____________________ Hex value in word labeled i1
stb    %l2, [%l0+3]
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
stb    %l2, [%l0+1]  _____________________ Hex value in word labeled i2
sra    %l2, 12, %l2  _____________________ Hex value in %l2
sth    %l2, [%l0+2]
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
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 ", bytes[i] );
    putchar( '\n' );
    return 0;
}

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

    for ( i = 1; i < size; i = i + 2 )
    {
        bytes[i] = (bytes[i-1] >> 4) | ((bytes[i] & 0xF) << 4) ;
        bytes[i-1] = (bytes[i] >> 4) | ((bytes[i-1] & 0xF) << 4) ;
    }
}
```

What is Rick's favorite beer? (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;

int main( int argc, char *argv[] ) {
    int b;
    int c = 42;

    foo( c, b );

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

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

    /* 6 */ (void) printf( "g --> %p\n", &g );
    /* 7 */ (void) printf( "d --> %p\n", &d );
    /* 8 */ (void) printf( "a --> %p\n", &a );
    /* 9 */ (void) printf( "f --> %p\n", &f );
    /* 10 */ (void) printf( "e --> %p\n", &e );
}
```
Extra Credit (10 points)
Optimize the following SPARC assembly program by filling in the delay slot nops with useful instructions starting after the clr instruction. (4 points)

```
.global main

.section ".rodata"
fmt: .asciz "%c"
.align 2
foo: .half 0x6465, 0x6B61, 0x4E20, 0x6672, 0x7553, 0x0000

.align 2
main:
save  %sp, -96, %sp
clr   %l1
set   foo, %l0
ldub  [%l0+%i0], %o1
tst   %o1
be    end
nop

loop:
inc   %l1
inc   %i0
set   fmt, %o0
call  printf, 2
nop

ldub  [%l0+%i0], %o1
tst   %o1
bne   loop

nop

derestores
```

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)

```
char a[] = "CSE30 Rocks!";
char *p = a + 3;

printf( "%c", ++*p );  _____
printf( "%c", ++++p );  _____
printf( "%c", *p++ + 1 );  _____
p++;
printf( "%c", *p = *p + 3 );  _____
printf( "%c", --*p++ );  _____
printf( "%d", p - a );  _____
```
<table>
<thead>
<tr>
<th>Character</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 NUL</td>
<td>01 SOH</td>
</tr>
<tr>
<td>08 BS</td>
<td>09 HT</td>
</tr>
<tr>
<td>10 DLE</td>
<td>11 DC1</td>
</tr>
<tr>
<td>18 CAN</td>
<td>19 EM</td>
</tr>
<tr>
<td>20 SP</td>
<td>21</td>
</tr>
<tr>
<td>28 (</td>
<td>29 )</td>
</tr>
<tr>
<td>30 0</td>
<td>31 1</td>
</tr>
<tr>
<td>38 8</td>
<td>39 9</td>
</tr>
<tr>
<td>40 @</td>
<td>41 A</td>
</tr>
<tr>
<td>48 H</td>
<td>49 I</td>
</tr>
<tr>
<td>50 P</td>
<td>51 Q</td>
</tr>
<tr>
<td>58 X</td>
<td>59 Y</td>
</tr>
<tr>
<td>60 `</td>
<td>61 a</td>
</tr>
<tr>
<td>68 h</td>
<td>69 i</td>
</tr>
<tr>
<td>70 p</td>
<td>71 q</td>
</tr>
<tr>
<td>78 x</td>
<td>79 y</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></td>
</tr>
<tr>
<td>&amp; address-of</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>