### CSE 30
**Fall 2012**
**Final Exam**

1. **Number Systems / C Compiling Sequence**
   
   ________________ (28 points)

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

3. **Branching**
   
   ________________ (16 points)

4. **Bit Operations**
   
   ________________ (13 points)

5. **Recursion/SPARC Assembly**
   
   ________________ (10 points)

6. **Local Variables, The Stack, Return Values**
   
   ________________ (19 points)

7. **More Recursive Subroutines**
   
   ________________ (10 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**
    
    ________________ (26 points)

**SubTotal**

_______________ (209 points)

**Extra Credit (5+%)**

_______________ (11 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 $FB6_{16}$ (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 $391_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

- sign-magnitude________________________________
- 1’s complement________________________________
- 2’s complement________________________________

Convert $-476_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

- sign-magnitude________________________________
- 1’s complement________________________________
- 2’s complement________________________________

C Compiling Sequence

Which part of the entire compilation sequence clear through to program execution is responsible for (6 points)

- translating assembly source code into object code ________________________________
- getting the executable image from disk into memory ________________________________
- creating an executable image from multiple object files ________________________________
- conditional compilation, macro expansion, and inclusion of header files ________________________________
- ensuring the bss segment is set up in memory and zero-filled ________________________________
- resolving undefined external symbols with defined global symbols across object modules __________________

Rt-Lt Rule

Using the C Rt-Lt Rule, define a variable named bar that is an array of 17 elements where each element is a pointer to a function that takes a single argument of type pointer to float and returns a pointer to an array of 9 elements where each element is of type pointer to struct foo. (7 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)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10110100</td>
<td>11110100</td>
<td>10110100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+10101101</td>
<td>+10110100</td>
<td>+01001100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101100</td>
<td>10110100</td>
<td>01001100</td>
<td></td>
</tr>
</tbody>
</table>

3. Branching
Given the following SPARC Assembly code write the equivalent function in C. Just perform a direct translation using the proper loop and conditional structure discussed in class/notes. Assume all local variables and return types are int. Name the local variable x. Use a while loop for the looping construct (not a for loop). (16 pts)

SPARC Assembly

.global foo
.section ".text"

foo:
save %sp, -(92 + 4) & -8, %sp
! 1 int local var named x
call rand ! Function prototype for rand: int rand(void);
nop
st %o0, [%fp - 4]
ld [%fp -4], %l0
cmp %l0, 99
bge L4
nop
L1:
ld [%fp - 4], %l1
cmp %l1, 42
bl L2
NOP
add %l1, 4, %l1
st %l1, [%fp - 4]
ba L3
nop
L2:
ld [%fp - 4], %o0
mov 5, %o1
call .rem
nop
st %o0, [%fp - 4]
L3:
ld [%fp - 4], %l2
cmp %l2, 99
bl L1
nop
L4:
ld [%fp - 4], %l3
add %l3, 17, %l3
st %l3, [%fp - 4]
ld [%fp - 4], %i0
ret
restore

C

```c

```
4. Bit Operations

What is the value of %l0 after each statement is executed? **Express your answers as 8 hexadecimal digits.**
(All 32 bits. Be sure to specify any leading or trailing zeros.)

```plaintext
set 0xCAFEFACE, %l0
set 0x86753099, %l1
and %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xCAFEFACE, %l0
sll %l0, 9, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xCAFEFACE, %l0
sra %l0, 10, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xCAFEFACE, %l0
set 0x????????, %l1
xor %l0, %l1, %l0  ! Value in %l0 is now 0x87654321

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

set 0xCAFEFACE, %l0
set 0x86753099, %l1
or %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xCAFEFACE, %l0
srl %l0, 11, %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
.global main
/* main.s */
.section "".rodata"
.align 4
code: .word 0x43216153, 0x21764567, 0x61336E4A, 0x306F2020, 0x4C265220, 0x2075652B
.word 0x6C762B65, 0x6943734C, 0x00000030
.global fubar
/* fubar.s */
.section "".rodata"
fmt: .asciz "%c"
.section "".text"
fubar:
.save %sp, -(92 + 1) & -8, %sp
inc %i1
cmp %i0, %g0
be end
nop
ldub [%i0 + %i1], %l0
cmp %l0, %g0
be end
nop
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
```

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( int x, char *y ) {
    struct foo {
        int   s1;
        short s2[7];
        char  s3;
        char *s4;
    }     local_stack_var1;
    int   *local_stack_var2;
    int   local_stack_var3;

    local_stack_var2 = &local_stack_var1.s1; /* 1 */
    local_stack_var1.s3 = *y + 7;  /* 2 */
    local_stack_var3 = local_stack_var1.s2[5] + x++; /* 3 */
    return ( x + ++*local_stack_var2 ); /* 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? (10 pts)

```c
#include <stdio.h>

int BBB( int );

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( "AAA = %d\n", AAA( 2 ) );
    return 0;
}
```

Put output here
8. Floating Point

Convert 156.875\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 0xC381A000 (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 + or - number of instructions away for the target (vs. a Label).

\texttt{lduh} [\%i2 + \%l5], \%o3 ____________________________________ (5 points)

\texttt{sll} \%l2, 15, \%o3 ____________________________________ (5 points)

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

0x3CBFFFF9 ____________________________________ (5 points)

0xE82FBFF5 ____________________________________ (5 points)
10. Linkage, Scope, Lifetime, Data

For the following program fragment, specify in which C runtime area/segment each symbol will be allocated or pointing: (32 points — 1 point each)

```c
static int a;

int b = 37;

static int c = 37;

int d;

static int foo( int e ) {
    int *f;
    f = (int *) malloc( e );
    static double g = 42.24;
    int (*h)(int) = foo;
    static int i;

    ...}
```

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 all 8 hex values requested after those lines have been fully executed. (11 points)
(All 32 bits. Be sure to specify any leading or trailing zeros.)

```
.global main

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

.c: .byte 0xCC
    .align 2

.s: .half 0xBEAD
    .align 4

.i1: .word 0x12345678
.i2: .word 0x12345678
.i3: .word 0x12345678
.x: .word 0x99990000

.section ".text"
main:
    save %sp, -96, %sp
    set x, %l0
    set s, %l1
    ldsh [%l1], %l2                      Hex value in %l2
    stb %l2, [%l0+2]                     Hex value in word labeled x
    srl %l2, 8, %l2                      Hex value in %l2
    stb %l2, [%l0]
    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]                     Hex value in word labeled i1
    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
    stb %l2, [%l0]                     Hex value in word labeled i2
    sra %l2, 12, %l2                      Hex value in %l2
    sth %l2, [%l0+2]                     Hex value in word labeled i2
    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

Optimize the following piece of SPARC Assembly to eliminate the nop instructions. Follow the optimization techniques used in class to fill the delay slots with useful instructions already in the code. (6 pts)

Just fill delay slots. Do not try to make sense of the code.

! Some code here - cannot use

L1:
    cmp   %l3, 42
    bg    L2
    nop
    add   %l3, 4, %l3
L2:
    cmp   %l3, 99
    ble   L1
    nop

! Some code here - cannot use

---

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 address/value to largest address/value. (2 points each)

```c
#include <stdio.h>
#include <stdlib.h>

void foo( int, int ); /* Function Prototype */

int main( int argc, char *argv[] )
{
    int a;
    static int b;
    int c = 37;
    foo( c, b );

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

void foo( int d, int e )
{
    static int f = 37;
    int g = 42;

    /* 5 */ (void) printf( "5: g --> %p\n", &g );
    /* 6 */ (void) printf( "6: foo --> %p\n", foo );
    /* 7 */ (void) printf( "7: d --> %p\n", &d );
    /* 8 */ (void) printf( "8: malloc --> %p\n", malloc(50) );
    /* 9 */ (void) printf( "9: f --> %p\n", &f );
    /* 10 */ (void) printf( "10: e --> %p\n", &e );
}
```

this line prints

```

_________

smallest value

_________

_________

_________

_________

_________

_________

_________

_________

_________

```

this line prints

```

_________

largest value

```

---

---
Extra Credit (11 points)

What is the value of each of the following expressions taken sequentially based on changes that may have been made in previous statements? (10 points)

```c
#include <stdio.h>

int main()
{
    char a[] = "CSE30 Rocks!";
    char *p = a + 4;

    printf( "%c", *p-- );
    printf( "%c", --*p );
    printf( "%c", ++++p );
    printf( "%c", *p++ + 1 );
    ++p;
    printf( "%c", *(p + 2) = *(p + 1) );
    p = p + 4;
    printf( "%c", p[-1] = *p + 2 );
    p--;
    printf( "%c", --*p++ );
    printf( "%d", p - a );
    printf( "%s", a );
    return 0;
}
```

What is Rick's favorite medicine? ________________________________ (1 point)
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</td>
<td>size of type/object</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