Login: cs30x_________  Name _________________________

Student ID ____________________  Signature_______________________

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
Winter 2013
Final Exam

1. Number Systems / C Compiling Sequence  _________________ (23 points)
2. Binary Addition/Condition Code Bits/Overflow Detection  _________________ (12 points)
3. Branching  _________________ (23 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  _________________ (9 points)
8. Floating Point  _________________ (12 points)
9. Machine Instructions  _________________ (20 points)
10. Linkage, Scope, Lifetime, Data  _________________ (34 points)
11. Load/Store/Memory  _________________ (11 points)
12. Miscellaneous  _________________ (28 points)

SubTotal  _________________ (214 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 $\text{FAFF}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

- **binary** __________________________________ (straight base conversion to binary)
- **octal** ________________________________ (straight base conversion)
- **decimal** ________________________________ (convert to signed decimal)

Convert $380_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

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

Convert $-465_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

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

Rt-Lt Rule

Using the C Rt-Lt Rule, define a variable named foo that is a pointer to an array of 42 elements where each element is a pointer to a function that takes a single argument of type pointer to char and returns a pointer to an array of 37 elements where each element is of type pointer to struct bar. (8 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)

<table>
<thead>
<tr>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10110100</td>
<td>10011111</td>
<td>00110100</td>
<td></td>
</tr>
<tr>
<td>+11101101</td>
<td>+10011111</td>
<td>+01001100</td>
<td></td>
</tr>
<tr>
<td></td>
<td></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></td>
<td></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></td>
<td></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></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Branching (23 points)
Translate the C code below into the equivalent unoptimized SPARC Assembly code. Just perform a direct translation – no optimizations. Use the local register mappings for the variables in assembly as specified.

C
/* Assume variables a and b have been properly declared as ints. */
for ( a = 0; a < b; ++a )
{
    if ( a < 900 )
    {
        a = a - b;
    }
    else
    {
        b = b % a;
    }
}

SPARC ASSEMBLY
! a is mapped to %12
! b is mapped to %15
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.)

```
set 0xDEADBEEF, %l0
set 0x13578642, %l1
and %l0, %l1, %l0

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

```
set 0xDEADBEEF, %l0
sll %l0, 11, %l0

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

```
set 0xDEADBEEF, %l0
sra %l0, 7, %l0

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

```
set 0xDEADBEEF, %l0
set 0x????????, %l1
xor %l0, %l1, %l0

Value in %l0 is now Ox13578642
Value set in %l1 must be this bit pattern ________________________________ (3 points)
```

```
set 0xDEADBEEF, %l0
set 0x13578642, %l1
or %l0, %l1, %l0

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

```
set 0xDEADBEEF, %l0
srl %l0, 9, %l0

Value in %l0 is ________________________________ (2 points)
```
5. Recursion/SPARC Assembly
Given `main.s` and `fubar.s`, what gets printed when executed? Yes … Draw stack frames! (10 points)

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

.main:
    save    %sp, -96, %sp
    mov     10, %o0
    call    fubar
    nop
    ret
    restore

.global fubar                   /* fubar.s */

.fubar:
    save    %sp, -(92 + 4) & -8, %sp
    cmp     %i0, %g0
    bnez    L1
    nop
    st      %i0, [%fp - 4]
    sub     %i0, 1, %o0
    call    fubar
    nop
    ld      [%fp - 4], %l0
    add     %o0, %l0, %l0
    set     fmt, %o0
    mov     %l0, %o1
    call    printf
    nop
    mov     %l0, %i0
    ba      L2
    nop
L1:
    mov     1, %i0
    ret
    restore
```

What gets printed? _________________________________________________________________________
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( int x, char *y ) {
    struct foo {
        int  s1;
        short s2[5];
        char s3;
        char *s4;
    } local_stack_var3;

    int *local_stack_var1;
    int  local_stack_var2;
    short s2[5];
    char s3;
    char *s4;

    local_stack_var1 = &local_stack_var3.s1; /* 1 */
    *y = local_stack_var3.s3++; /* 2 */
    local_stack_var2 = ++x + local_stack_var3.s2[1]; /* 3 */

    return ( *++local_stack_var1 + 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? (9 pts)

```c
#include <stdio.h>
#define SIZE 3

void ruler( int );

int main()
{
    int n = SIZE;
    int i;

    for ( i = 0; i <= n; ++i )
        printf( "=" ); /* no newline */
    printf( "\n" );   /* newline */
    ruler( n );

    for ( i = 0; i <= n; ++i )
        printf( "=" ); /* no newline */
    printf( "\n" );   /* newline */
    return 0;
}

void
ruler( int n )
{
    int i;

    if ( n > 0 )
    {
        ruler( n-1 );

        printf( "%d", n );      /* no newline */

        for ( i = 0; i < n; ++i )
            printf( "-" );        /* no newline */

        printf( "\n" );         /* newline */
        ruler( n-1 );
    }
}
```

Put output here
8. Floating Point

Convert $142.875_{10}$ (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

<table>
<thead>
<tr>
<th>Representation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary fixed-point</td>
<td>___________________________</td>
</tr>
<tr>
<td>IEEE floating-point</td>
<td>___________________________</td>
</tr>
</tbody>
</table>

Convert $0xC344C000$ (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).

- `stb    %l3, [%fp - l5]`  
  | | (5 points) |
- `addcc  %l5, %i1, %o2`  
  | | (5 points) |

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

- `0xD6164012`  
  | | (5 points) |
- `0x24800011`  
  | | (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: (34 points — 1 point each)

static int a;  
static int b = 37;  
int c = 37;  
int d;  
int foo( int e ) {  
    double f = 42.24;  
    static int *g = (int *) malloc( e );  
    static int h;  
    int (*i)(int) = foo;  
...
}

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 _______

foo _______  
a _______  
b _______  
c _______  
d _______  
e _______  
f _______  
g _______  
h _______  
i _______

If the function foo() above is called 5 times, indicate how many times will \( f \) be initialized?  
If the function foo() above is called 5 times, indicate how many times will \( g \) be initialized?  

9
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 0xA9

.s:
.align 4
i1: .word 0x915ACDC9
i2: .word 0x915ACDC9
i3: .word 0x915ACDC9
x: .word 0x00005678

.section ".text"
main:

save %sp, -96, %sp

set  x, %10
set  s, %11
lduh [%11], %12                      Hex value in %12

stb %12, [%10+2]                       Hex value in word labeled x

sll %12, 4, %12                       Hex value in %12
stb %12, [%10]

set   fmt, %o0
ld    [%10], %o1
call  printf                      Hex value in word labeled x
nop          (same as output of this printf)

set  i1, %10
set  c, %11
ldsb   [%11], %12                   Hex value in %12

sth %12, [%10]
stb %12, [%10+2]

set   fmt, %o0
ld    [%10], %o1
call  printf                      Hex value in word labeled i1
nop          (same as output of this printf)

set  i2, %10
set  i3, %11
ld    [%11], %12                   Hex value in %12

sra %12, 8, %12                     Hex value in %12
sth %12, [%10+2]

set   fmt, %o0
ld    [%10], %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? (Hint: Draw stack frames!)

```c
#include <stdio.h>
#include <stdlib.h>
void foo( int, int );   /* Function Prototype */
static int a;
int main( int argc, char *argv[] )
{
    int b = 7;
    static int c = 11;
    foo( argc, c );
    /* 1 */ (void) printf( "1: argc --> %p\n", &argc );
    /* 2 */ (void) printf( "2: malloc --> %p\n", malloc(50) );
    /* 3 */ (void) printf( "3: foo --> %p\n", foo );
    /* 4 */ (void) printf( "4: b --> %p\n", &b );
    /* 5 */ (void) printf( "5: c --> %p\n", &c );
    void foo( int d, int e )
    { int tmp;
      int tmp = a;
      *a = *b;
      *b = tmp;
    }
}
```

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
void swap1( int a, int *b )
{
  int tmp;
  tmp = a;
  a = *b;
  *b = tmp;
}

void swap2( int *a, int *b )
{
  int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}

int swap4( int *a, int b )
{
  int tmp;
  tmp = *a;
  *a = b;
  b = tmp;
  return b;
}
```

```c
int main()
{
  int a = 3;
  int b = 8;
  swap1( a, &b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 6;
  b = 12;
  swap2( &a, &b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 4;
  b = 7;
  swap3( a, b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 9;
  b = 5;
  b = swap4( &a, b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  return 0;
}
```

Put answer here

```c
int main()
{
  int a = 3;
  int b = 8;
  swap1( a, &b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 6;
  b = 12;
  swap2( &a, &b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 4;
  b = 7;
  swap3( a, b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 9;
  b = 5;
  b = swap4( &a, b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  return 0;
}
```

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

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

static int a;

int main( int argc, char *argv[] )
{
    int b = 7;
    static int c = 11;
    foo( argc, c );
    /* 1 */ (void) printf( "1: argc --> %p\n", &argc );
    /* 2 */ (void) printf( "2: malloc --> %p\n", malloc(50) );
    /* 3 */ (void) printf( "3: foo --> %p\n", foo );
    /* 4 */ (void) printf( "4: b --> %p\n", &b );
    /* 5 */ (void) printf( "5: c --> %p\n", &c );
}
```

```c
void swap1( int a, int *b )
{
  int tmp;
  tmp = a;
  a = *b;
  *b = tmp;
}

void swap2( int *a, int *b )
{
  int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}

int swap4( int *a, int b )
{
  int tmp;
  tmp = *a;
  *a = b;
  b = tmp;
  return b;
}
```

Put answer here

```c
int main()
{
  int a = 3;
  int b = 8;
  swap1( a, &b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 6;
  b = 12;
  swap2( &a, &b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 4;
  b = 7;
  swap3( a, b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  a = 9;
  b = 5;
  b = swap4( &a, b );
  printf( "%d\n", a );
  printf( "%d\n", b );
  return 0;
}
```
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[] = "Rock Me CSE30";
    char *p = a + 5;
    printf( "%c", *p++ );

    printf( "%c", --*p );
    printf( "%c", ++++p );
    printf( "%c", *p++ + 3 );

    ++p;
    printf( "%c", *(p + 2) = *(p + 1) );

    p = p + 2;
    printf( "%c", p[-1] = *p + 3 );

    p--;
    printf( "%c", --p++ );
    printf( "%d", p - a );
    printf( "%s", a );

    return 0;
}
```

What is Rick's rapper name? ___________________________________________ (1 point)
### Hexadecimal - Character

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

---

* indirection | R to L
++ prefix increment
-- prefix decrement
& address-of
sizeof size of type/object
(type) type cast
---

* multiplication | L to R
/ division
% modulus
---

+ addition | L to R
- subtraction
---

= assignment | R to L