CSE 30
Winter 2012
Final Exam

1. Number Systems / C Compiling Sequence
   _________________________ (15 points)

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

3. Branching
   _________________________ (24 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
    _________________________ (25 points)

SubTotal (100%) _________________________ (206 points)

Extra Credit _________________________ (15 points)

Total _________________________
1. Number Systems
Convert 0xFB7B (2’s complement, 16-bit word) to the following. (6 points)

- **binary**
  
- **octal**
  
- **decimal** (convert to signed decimal)

Convert \(288_{10}\) to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

- **sign-magnitude**

- **1’s complement**

- **2’s complement**

Convert \(-387_{10}\) to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

- **sign-magnitude**

- **1’s complement**

- **2’s complement**

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)

\[
\begin{align*}
11010111 & \quad +00101001 & \quad 10101010 & \quad +01010101 \\
\hline
\end{align*}
\]

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

2
3. **Branching** (24 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 < 9000 )
    {
        a = a - b;
    }
    else
    {
        b = b / a;
    }
}
```

```sparc
! 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 in hexadecimal.**

```c
set 0xC0FFEE69, %l0
set 0xBABEFACE, %l1
and %l0, %l1, %l0

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

```c
set 0xC0FFEE69, %l0
sra %l0, 9, %l0

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

```c
set 0xC0FFEE69, %l0
sll %l0, 7, %l0

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

```c
set 0xC0FFEE69, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0

! Value in %l0 is now 0xBABEFACE

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

```c
set 0xC0FFEE69, %l0
set 0xBABEFACE, %l1
or %l0, %l1, %l0

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

```c
set 0xC0FFEE69, %l0
srl %l0, 8, %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 */
.align 4
.rodata
  .word 0x43736453, 0x72654574, 0x6B335061, 0x303D4E20, 0x2B20522B, 0x66754372
  .word 0x6C2F7565, 0x43537300, 0x00000030
.text
main:
  save  %sp, -92 & -8, %sp
  set   code, %o0
  mov   0, %o1
  call  fubar
  nop
  ret
  restore

.global fubar  /* fubar */
.rodata
  .ascii "%c"
.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
```

Output:
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. (20 points)

```c
int fubar( int a, short b ) {
    struct foobaz {
        int s1[3];
        short s2;
        long s3;
        char s4[4];
    } local_stack_var1;
    int local_stack_var2;
    int * local_stack_var3;
    local_stack_var1.s4[2] = local_stack_var1.s4[0]; /* 1 */
    local_stack_var2 = ++a + local_stack_var1.s1[0]; /* 2 */
    local_stack_var1.s2 = b - local_stack_var1.s3; /* 3 */
    return *local_stack_var3++ + a; /* 4 */
}
```

Write the equivalent full unoptimized SPARC assembly language module. **You must allocate all local variables on the stack.** No short cuts. Treat each statement independently. Label & draw lines between stmts.
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", AAA( 10 ) );
    return 0;
}
```

Put Answer Here
8. Floating Point

Convert 120.875_{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 0xC3702000 (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).

\texttt{addcc} %l3, -9, %i2  _____________________________________  (5 points)
\texttt{sth} %o2, [%i3 + %l1]  _____________________________________  (5 points)

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

\texttt{0x2E800008}  _____________________________________  (5 points)
\texttt{0xAC1AC019}  _____________________________________  (5 points)
10. Linkage, Scope, Lifetime, Data

For the following program fragment, specify what C runtime area/segment will be used for each symbol definition or statement: (32 points — 1 point each)

```c
int a;

static int b = 42;
static int c;
int d = -99;

int foo(int e) {
    double f = 42.420;
    int g;
    static int *h;
    h = (int *) malloc(e);
    static 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) Visible just to this source file.
C) Local to function foo().

```c
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.

```c
foo _______ 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 0xBE

.align 2
s: .half 0xCafe

.align 4
i1: .word 0x87654321
i2: .word 0x87654321
i3: .word 0x87654321
x: .word 0x77007700

.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]
  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 character gets printed with each putchar() line?

```c
unsigned int x = 0xA613;

putchar( (((x << 8) & 0xFFFF) >> 12) + '0' );

putchar( (x & 0xF) + '0' );

putchar( (x ^ x) + '1' );

putchar( '0' - ((x & 0xF00)>> 8) );
```

Give the order of the typical C compilation stages and on to actual execution as discussed in class

A – Object file (prog.o)    B – Assembly file (prog.s)
C – Loader    D – ccomp (C compiler)
E – as (Assember)    F – Source file (prog.c)
G – Program Execution    H – ld (Linkage Editor)
I – prog.exe/a.out (Executable image)    J – cpp (C preprocessor)
K – Segmentation Fault (Core Dump) / General Protection Fault

gcc ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____

Given the following program, specify the order of the output lines when run and sorted by the address printed with the %p format specifier on a Sun SPARC Unix and Linux system. For example, which line will print the lowest memory address, then the next higher memory address, etc. up to the highest memory address?

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

int a = 42;

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

    foo( c, b );

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

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

    /* 6 */ (void) printf( "d --> %p\n", &d );
    /* 7 */ (void) printf( "g --> %p\n", &g );
    /* 8 */ (void) printf( "foo --> %p\n", foo );
    /* 9 */ (void) printf( "f --> %p\n", &f );
    /* 10 */ (void) printf( "e --> %p\n", &e );
}
```
Extra Credit
What gets printed when this program is executed?

```c
#include <stdio.h>

int main()
{
    char a[] = "CSE30";
    char *p = a;

    printf( "%c\n", *p++ );  
    printf( "%c\n", *(p+3) = *p);  
    printf( "%c\n", *++p );  
    printf( "%c\n", --*p++ );  
    printf( "%c\n", +++p );  
    printf( "%d\n", ++p - a );  
    printf( "%s\n", a );  

    return 0;
}
```

What is Rick's password?  

Using the Right-Left rule write the C definition of a variable named foo that is a pointer to an array of 3 elements where each element is a pointer to a function that takes a single argument of type pointer to short and returns a pointer to a struct construct.
### 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 DEL</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>
<tr>
<td>*</td>
<td>indirection</td>
</tr>
<tr>
<td>++</td>
<td>prefix increment</td>
</tr>
<tr>
<td>--</td>
<td>prefix decrement</td>
</tr>
<tr>
<td>&amp;</td>
<td>address-of</td>
</tr>
<tr>
<td>sizeof</td>
<td>size of type/object</td>
</tr>
<tr>
<td>(type)</td>
<td>type cast</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
</tbody>
</table>
Scratch Paper