CSE 30
Spring 2010
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
    ___________________ (21 points)

SubTotal (100%) ___________________ (202 points)

Extra Credit ___________________ (18 points)

Total ___________________
1. Number Systems

Convert \(0xFFFF\) (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 \(299_{10}\) to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

- Sign-magnitude: ________________________________
- 1’s complement: ________________________________
- 2’s complement: ________________________________

Convert \(-398_{10}\) 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)

\[
\begin{array}{c}
11010110 \\
+00111011
\end{array}
\begin{array}{c}
10110100 \\
+11001100
\end{array}
\begin{array}{c}
01011101 \\
+00101001
\end{array}
\]

<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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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. Remember to implement short-circuiting logic for the logical OR.

```
/* Assume variables a and b have been properly declared as ints. */
for ( a = 0; a < b; ++a )
{
    if ( b < 9000 )
    {
        b = b * a;
    }
    else
    {
        a = a + b;
    }
}
```

```
/* a is mapped to %l3 
 ! b is mapped to %l6
```

4. Bit Operations

What is the value of %l0 after each statement is executed? **Express your answers in hexadecimal.**

```
set 0xDEADBEEF, %l0
set 0x2468ACE1, %l1
and %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, 7, %l0
Value in %l0 is ____________________________ (2 points)
```

```
set 0xDEADBEEF, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0                         ! Value in %l0 is now Ox2468ACE1
Value set in %l1 must be this bit pattern ____________________________ (3 points)
```

```
set 0xDEADBEEF, %l0
set 0x2468ACE1, %l1
or %l0, %l1, %l0
Value in %l0 is ____________________________ (2 points)
```

```
set 0xDEADBEEF, %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 */
.section "".rodata"
.align 4
code: .word 0x43216153, 0x21764567, 0x61336E4A, 0x306F2020, 0x4C3E5220, 0x20756543
       .word 0x6C760065, 0x6900734C, 0x00000030
.main:  save %sp, -92 & -8, %sp
       set code, %o0
       mov 1, %o1
       call fubar
       nop
       ret
       restore

.global fubar    /* fubar */

.fmt: .asciz "%c"

.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:
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. (20 points)

```c
int fubar( int a, short b ) {
    int local_stack_var1;
    int * local_stack_var2;
    struct foobaz {
        int s1[3];
        char s2;
        long s3;
        char s4[8];
    } local_stack_var3;

    local_stack_var3.s4[5] = 'X';                      /* 1 */
    local_stack_var1 = ++b + local_stack_var3.s1[1];   /* 2 */
    local_stack_var3.s2 = local_stack_var3.s4[2] - a;  /* 3 */
    return *local_stack_var2++ + 42;                   /* 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 x )
{
    int result;
    printf( "x = %d\n", x );
    if ( x <= 1 )
        return 2;
    else
    {
        result = BBB( x - 1 );
        printf( "result = %d\n", result + x + 2 );
        return result;
    }
}

int
BBB( int x )
{
    int result;
    printf( "x = %d\n", x );
    if ( x <= 1 )
        return 1;
    else
    {
        result = AAA( x - 2 );
        printf( "result = %d\n", result + x + 1 );
        return result;
    }
}

int
main( int argc, char *argv[] )
{
    printf( "%d\n", BBB( 8 ) );
    return 0;
}
```

Put Answer Here
8. Floating Point

Convert $151.375_{10}$ (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 $0xC353E000$ (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).

ldsh [%l2 + %o5], %i3 __________________________________ (5 points)
sll %l4, 15, %o1 __________________________________ (5 points)

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

0x3CBFFFF8 __________________________________ (5 points)
0xD62FBFF7 __________________________________ (5 points)
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
static int a;
int b = 42;
int c;
static int d = -99;
static int foo( int e ) {
    static double f = 42.24;
    static int g;
    int (*h)(int) = foo;
    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) Visible just to this source file. C) Local to function foo().

```plaintext
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.

```plaintext
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 0XXXXXXXX

c: .byte 0x97
.s: .half 0xFACE

.i1: .word 0x2468ACE1
.i2: .word 0x2468ACE1
.i3: .word 0x2468ACE1
.x: .word 0xBABE0000

.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
    sll  %l2, 8, %l2                     Hex value in %l2
    sth  %l2, [%l0+2]

    set  fmt, %o0
    ld  [%l0], %o1
    call printf
    nop                                       Hex value in word labeled x
                                                (same as output of this printf)

    set  i1, %l0
    set  c, %l1
    ldsb  [%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
    nop                                       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

    stb  %l2, [%l0+2]                     Hex value in word labeled i2
    sra  %l2, 8, %l2                     Hex value in %l2
    sth  %l2, [%l0]

    set  fmt, %o0
    ld  [%l0], %o1
    call printf
    nop                                       Hex value in word labeled i2
                                                (same as output of this printf)

    ret
    restore
12. Miscellaneous

List the five steps in the instruction cycle as discussed in class. Yes, order is important.

1) _____________________________
2) _____________________________
3) _____________________________
4) _____________________________
5) _____________________________

Order by general access speed the following data storage types. [ 1 is fastest, 6 is slowest ]

SASD _____ (Sequential Access Storage Device) Tape Drive, for example
L1 cache _____
RAM _____
DASD _____ (Direct Access Storage Device) Disk Drive, for example
L2 cache _____
registers _____

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;
int main( int argc, char *argv[] ) {
  int b;
  static 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( "foo --> %p\n", foo );
  /* 5 */ (void) printf( "b --> %p\n", &b );
}

void foo( int d, int e ) {
  int f = 42;
  int g;
  /* 6 */ (void) printf( "d --> %p\n", &d );
  /* 7 */ (void) printf( "g --> %p\n", &g );
  /* 8 */ (void) printf( "a --> %p\n", &a );
  /* 9 */ (void) printf( "f --> %p\n", &f );
  /* 10 */ (void) printf( "e --> %p\n", &e );
}
```

prints smallest value
prints largest value
Extra Credit
What gets printed when this program is executed?

```c
#include <stdio.h>

int main()
{
    char a[] = "diversion">
char b[] = "sound">
char *p1 = a;
char *p2 = b;

    printf( "%c", ++*p1 );
    printf( "%c", a[strlen(a) - 1] - 2 );
    printf( "%c", *(b + 4) - 1 );
    printf( "%c", --*p2 );
    printf( "%c", *++p2 );
    printf( "%c", ++*p2 );
    printf( "%c", p2[-1] + 1 );

    /* BTW, Diversion Sound is Kellen Steffen's band. */
    printf( "%c", pl[2] + 3 );
    printf( "%c", *p1++ - strlen(p2) );
    printf( "%c", *p1 + 2 );
    printf( "%c", *p2++ );
    return 0;
}
```

Using the Right-Left rule write the C definition of a variable named fubar that is a pointer to a function that takes a pointer to a char and returns a pointer to an array of 7 elements where each element is a pointer to a struct Sporcle.

What is the name of Rick's dog? _________________________________
### 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>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>*</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