CSE 30
Winter 2009
Final Exam

1. Number Systems / C Compiling Sequence ___________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________ (12 points)
3. Branching ___________________ (20 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 ___________________ (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 ___________________ (27 points)

SubTotal ___________________ (202 points)
Extra Credit ___________________ (15 points)
Total ___________________
1. Number Systems

Convert \(0xB76\) (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 \(429_{10}\) to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

sign-magnitude_____________________________________________

1’s complement_____________________________________________

2’s complement_____________________________________________

Convert \(-392_{10}\) 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)

\[
\begin{array}{c}
11010110 \\
+10111011 \\
\hline
\end{array}
\quad
\begin{array}{c}
10111001 \\
+01010110 \\
\hline
\end{array}
\quad
\begin{array}{c}
01011101 \\
+01101001 \\
\hline
\end{array}
\]

\[
\begin{array}{cccc}
N & Z & V & C \\
\hline
| | | | |
\end{array}
\quad
\begin{array}{cccc}
N & Z & V & C \\
\hline
| | | | |
\end{array}
\quad
\begin{array}{cccc}
N & Z & V & C \\
\hline
| | | | |
\end{array}
\]
3. Branching (20 points)
Given the following SPARC assembly code write the equivalent function in C. Use only standard loops and if/else statements; no gotos!

```
SPARC ASSEMBLY
.global branching
.section ".text"
branching:
save  %sp, -96, %sp
call  rand
nop
mov  %o0, %l1     ! local var int x
! mapped to %l1
cmp  %l1, 99
bge  L4
nop
L1:
cmp  %l1, 42
bl   L2
nop
add  %l1, 4, %l1
ba   L3
nop
L2:
mov  %l1, %o0
mov  5, %o1
call  .rem
nop
mov  %o0, %l1
L3:
cmp  %l1, 99
bl   L1
nop
L4:
add  %l1, 17, %l1
mov  %l1, %i0
ret
restore
```

C

```
for the following instruction sequence, mark with an X under the conditional branch instructions which would transfer control to doit if used in place of ba. [+1 correct; -1 incorrect]
```

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>blu</th>
<th>bpos</th>
<th>bgu</th>
<th>bne</th>
<th>ble</th>
<th>bge</th>
<th>bneg</th>
<th>bvs</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov -5, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmp %10, 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba doit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Bit Operations

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

set 0xFACECAFE, %l0
set 0x86715309, %l1
and %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFACECAFE, %l0
sra %l0, 14, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFACECAFE, %l0
sll %l0, 13, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFACECAFE, %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 0xFACECAFE, %l0
set 0x86715309, %l1
or %l0, %l1, %l0

Value in %l0 is _______________________________________ (2 points)

set 0xFACECAFE, %l0
srl %l0, 10, %l0

Value in %l0 is _______________________________________ (2 points)
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

section ".text"
main:
save   %sp, -92 & -8, %sp
set    code, %o0
mov    1, %o1
call   fubar
nop

ret
restore

.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. (20 points)

```c
long fubar( char *x, int y ) {
    int  *local_stack_var1;
    struct foo {
        short s1[5];
        char  s2;
        char *s3;
        int   s4;
    }        local_stack_var2;
    int   local_stack_var3;
    local_stack_var2.s2 = *x + 5; /* 1 */
    local_stack_var3 = local_stack_var2.s1[2] + ++y; /* 2 */
    local_stack_var1 = &local_stack_var2.s4; /* 3 */
    return ( y + *local_stack_var1++ ); /* 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.
7. More Recursive Subroutines / Drawing Stack Frames

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 165.75\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 0xC356E000 (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 \texttt{hexadecimal} values for your answers. If an instruction is a branch, specify the number of instructions away for the target (vs. a Label).

\texttt{ldub} [%i5 + %l4], %o2 __________________________________ (5 points)
\texttt{sub} %l3, 15, %i4 __________________________________    (5 points)

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

\texttt{0xD43FBFF8} __________________________________    (5 points)
\texttt{0x3EBFFFF7} __________________________________    (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;

int b = 37;

static int c;

static int d = 88;

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

foo _______  foo _______

a _______  a _______
b _______  b _______
c _______  c _______
d _______  d _______
e _______  e _______
f _______  f _______
g _______  g _______
h _______  h _______
i _______  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.
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 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+1]                     Hex value in word labeled x
srl %l2, 12, %l2                    Hex value in %l2
stb %l2, [%l0+3]

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
ldub [%l1], %l2                      Hex value in %l2
stb %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]                     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
nop                                     Hex value in word labeled i2
(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.

L1:
    cmp     %l1, 42
    bl      L2
    nop
    add     %l1, 4, %l1
L2:
    cmp     %l1, 99
    bl      L1
    nop

What is Rick's favorite drink in the morning? (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 = 24;
int main( int argc, char *argv[] ) {
    int b = 99;
    int c;
    foo( c, b );
    /* 1 */ (void) printf( "c --> %p\n", &c );
    /* 2 */ (void) printf( "foo --> %p\n", foo );
    /* 3 */ (void) printf( "b --> %p\n", &b );
    /* 4 */ (void) printf( "argc --> %p\n", &argc );
    /* 5 */ (void) printf( "malloc --> %p\n", malloc(50) );
}
void foo( int d, int e ) {
    int f = 42;
    static int g;
    /* 6 */ (void) printf( "g --> %p\n", &g );
    /* 7 */ (void) printf( "e --> %p\n", &e );
    /* 8 */ (void) printf( "a --> %p\n", &a );
    /* 9 */ (void) printf( "f --> %p\n", &f );
    /* prints smallest value */
    /* prints largest value */
```
/* 10 */ (void) printf( "d --> %p\n", &d );
}

Extra Credit
What is the output of the following program? (8 points)

#include <stdio.h>
#define SIZE 4
void mystery( unsigned short hwords[], size_t size );
int
main()
{
    int i;
    unsigned short hwords[SIZE] = { 0x1942, 0xF837, 0x13AC, 0xB6D5 };  
mystery( hwords, SIZE );
    for ( i = 0; i < SIZE; ++i )
        {  
            printf( "0x%04X\n", hwords[i] ); /* Prints 0xXXXX */
        }
    return 0;
}
void
mystery( unsigned short hwords[], size_t size )
{
    int i, j;
    for ( i = 0; i < size; ++i )
    {
        if ( i & 1 )
        {
            j = i * 4;
            hwords[i] = (hwords[i] << j) | (hwords[i] >> (16 - j));
        }
        else
        {
            j = (i + 1) * 4;
            hwords[i] = (hwords[i] >> j) | (hwords[i] << (16 - j));
        }
    }
}

What is the value of each of the following expressions taken sequentially based on changes that may have been made in previous statements? (7 points)

char a[] = "Hang 10!";
char *p = a;
printf( "%c", *p = toupper( ++p[1] ) );   _____
printf( "%c", *++p = a[strlen(a) - 6] - 2 );   _____
printf( "%c", *p++ + 1 );  _____
p = p + 3;
printf( "%c", *p = *p + 3 );   _____
printf( "%c", --*p++ );  _____
printf( "%d", p - a );  _____
printf( "%s", a );  _______________________.

12
### 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>-----------------</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>
Scratch Paper
Scratch Paper