CSE 30
Spring 2007
Final Exam

1. Number Systems ___________________  (25 points)

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

3. Branching ___________________  (19 points)

4. Bit Operations ___________________  (13 points)

5. Recursion/SPARC Assembly
   ___________________  (10 points)

6. Local Variables, The Stack, Return Values
   ___________________  (22 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
    ___________________  (9 points)

12. Miscellaneous
    ___________________  (29 points)

SubTotal ___________________  (215 points)

Extra Credit ___________________  (10 points)

Total ___________________
1. Number Systems

Convert $\text{FB3}_{16}$ (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 $\text{415}_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

**sign-magnitude**__________________________________

**1’s complement**________________________________

**2’s complement**________________________________

Convert $\text{-396}_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

**sign-magnitude**__________________________________

**1’s complement**________________________________

**2’s complement**________________________________

C Compiling Sequence

Put the following in the correct order/sequence using the numbers to the left of each word: (10 points)

1. loader    5. program execution    9. resulting .s file
2. executable (.exe/a.out)  6. assembler    10. resulting .o file
3. compiler    7. preprocessor
4. source code (.c file)  8. linkage editor

_____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____ —> _____
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)

\[
\begin{align*}
00101100 & \quad +11010100 \\
+01110110 & \quad +01001101 \\
--------- & \quad --------- \\
10101010 & \quad +11010110
\end{align*}
\]

\[
\begin{array}{cccccc}
N & Z & V & C \\
\hline
\quad & \quad & \quad & \quad & \quad & \quad \\
\quad & \quad & \quad & \quad & \quad & \quad \\
\quad & \quad & \quad & \quad & \quad & \quad
\end{array}
\]

3. Branching

Given the following SPARC assembly code write the equivalent function in C. Use only standard loops and if/else statements; no gotos! (19 points)

```
SPARC ASSEMBLY                   C
.global subtract
.section " .text"
subtract:
    save %sp, -96, %sp
    mov %g0, %l0        ! local var count
                      ! mapped to %l0
    cmp %i0, %i1
    be L3
    nop
    bl L2
    nop
L1:
    inc %i1
    inc %i0
    cmp %i0, %i1
    bne L1
    nop
    mov %i0, %i0
    ret
    restore
L2:
    inc %i0
    inc %i0
    cmp %i0, %i1
    bne L2
    nop
    mov %i0, %i0
    neg %i0
    ret
    restore
L3:
    mov %g0, %i0
    ret
    restore
```
4. Bit Operations

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

```
set 0xDB3724AC, %l0
set 0xA8675309, %l1
or  %l0, %l1, %l0

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

```
set 0xDB3724AC, %l0
srl  %l0, 13, %l0

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

```
set 0xDB3724AC, %l0
sll  %l0, 9, %l0

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

```
set 0xDB3724AC, %l0
set 0x????????, %l1
btog %l1, %l0

! Value in %l0 is now OxCAFEBABE

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

```
set 0xDB3724AC, %l0
set 0xA8675309, %l1
and  %l0, %l1, %l0

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

```
set 0xDB3724AC, %l0
sra  %l0, 7, %l0

Value in %l0 is _______________________________________ (2 points)
```
5. Recursion/SPARC Assembly
Given main.s and recurse.s, what gets printed when executed? (10 points)

```
.global main /* main.s */
.section "text"
main:
save  %sp, -96, %sp
mov   3, %0
mov   2, %1
call  recurse
nop
ret
restore

.global recurse /* recurse.s */
.section "data"
fmt:   .asciz "%d
"
.section "text"
recurse:
save  %sp, -(92 + 8) & -8, %sp     ! 2 local int variables
sub   %0, %1, %l0
st    %l0, [%fp - 4]
set   fmt, %0
mov   %l0, %1
call  printf
nop
cmp   %0, 8
bge   L1
nop
add  %0, 2, %0
mov   %l1, %0
call  recurse
nop
ld    [%fp - 4], %l0
add  %l0, %0, %l1
st    %l1, [%fp - 8]
ba    L2
nop
L1:
st    %l0, [%fp - 8]
L2:
set   fmt, %0
ld    [%fp - 8], %1
call  printf
nop
ld    [%fp - 8], %0
ret
restore
```
int fubar( char x, short y ) {
    int *local_stack_var1;
    struct foo {
        char s1[3];
        short s2;
        int s3[3];
    } local_stack_var2;
    char *local_stack_var3;
    local_stack_var2.s2 = y + *local_stack_var1++; /* 1 */
    local_stack_var2.s1[1] = *local_stack_var3 + x; /* 2 */
    *local_stack_var1 = local_stack_var2.s3[2] - y; /* 3 */
    return ( local_stack_var2.s2 + 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. (22 points)
7. More Recursive Subroutines

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

int BBB( int x )
{
    int result;

    printf( "x = %d\n", x );

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

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

Put output here
8. Floating Point

Convert $136.625_{10}$ (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 $0xC34CC000$ (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).

```
addcc %l2, -9, %i1
___________________________________  (5 points)
```

```
sth  %o2, [%l3 + %g5]
___________________________________  (5 points)
```

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

```
0x24800008
___________________________________  (5 points)
0xBC1CC009
___________________________________  (5 points)
```
10. Linkage, Scope, Lifetime, Data

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

static int a;
int b = 42;
int c;
static int d = 404;
static int foo( int e ) {
    static double f = 4.20;
    int g = 8675309;
    static int *h;
    h = (int *) malloc( b );
    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 _______ i _______ h _______ g _______ f _______ e _______ d _______ c _______ b _______ a _______

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 _______ i _______ h _______ g _______ f _______ e _______ d _______ c _______ b _______ a _______
11. Load/Store/Memory
What gets printed in the following program? (9 points)

.global main

.section ".data"
fmt: .asciz "0x%08X\n" ! prints value as hex 0XXXXXXXXX
c: .byte 0x88
.s: .half 0xCAFE
.il: .word 0x9ABCD123
.i2: .word 0x9ABCD123
.i3: .word 0x9ABCD123
.x: .word 0x00004444

.section ".text"
main:
save %sp, -96, %sp
.set il, %10
.set s, %11
.ldsh [%11], %11
.sth %11, [%10+2]
.stb %11, [%10+1]
.set fmt, %o0
.ld [%10], %o1
.call printf
nop

.set i2, %10
.set c, %11
.ldsb [%11], %11
.sth %11, [%10]
.stb %11, [%10+3]
.set fmt, %o0
.ld [%10], %o1
.call printf
nop

.set x, %11
.set i3, %10
.ldsb [%11+2], %12
.sth %12, [%10]
.stb %12, [%10+2]
.set fmt, %o0
.ld [%10], %o1
.call printf
nop

ret
restore
12. Miscellaneous
What is the output of the following program? (8 points)

```c
#include <stdio.h>
#define SIZE 4
char * mystery( unsigned char bytes[], size_t size );

int main() {
    unsigned char bytes[SIZE] = { 0x19, 0x42, 0x08, 0x37 }; 
    printf( "%s\n", mystery( bytes, SIZE ) );
    return 0;
}

char * mystery( unsigned char bytes[], size_t size )
{
    int i;
    int j = 0;
    char * str = (char *) malloc ((size * 2) + 1);
    for ( i = 0; i < size; ++i ) {
        str[j++] = (bytes[i] >> 4) + '0';
        str[j++] = (bytes[i] & 0xF) + '0';
    }
    str[j] = '\0';
    return str;
}
```

Give the equivalent C expression for the following without using array access square brackets ([ ]). (1 pt)

```c
a[3] _____________
```

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 */
static int a;

int main( int argc, char *argv[] ) {
    int b;
    int c = 69;
    foo( argc, c );

    /* 1 */ (void) printf( "b --> %p\n", &b );
    /* 2 */ (void) printf( "malloc --> %p\n", malloc(50) );
    /* 3 */ (void) printf( "foo --> %p\n", foo );
    /* 4 */ (void) printf( "a --> %p\n", &a );
    /* 5 */ (void) printf( "argv --> %p\n", &argv );
    /* 6 */ (void) printf( "c --> %p\n", &c );
}

void foo( int d, int e ) {
    int f = e;
    static int g = 5;
    /* 7 */ (void) printf( "g --> %p\n", &g );
    /* 8 */ (void) printf( "e --> %p\n", &e );
    /* 9 */ (void) printf( "f --> %p\n", &f );
    /* 10 */ (void) printf( "d --> %p\n", &d );
}
```

Put output here
Extra Credit (10 points)
Consider the following SPARC assembly program

```
.global main

.section ".rodata"
fmt: .asciz "%c"
.align 2
foo: .half 0x6465, 0x6B61, 0x4E20, 0x6672, 0x7553, 0x0000

.section ".text"
main:
save %sp, -96, %sp
clr %l1
set foo, %l0
ldub [%l0+%l1], %o1
tst %o1 ! cmp %o1, %g0
be end
nop
loop:
set fmt, %o0
ldub [%l0+%l1], %o1
call printf, 2
nop
inc %l1
ldub [%l0+%l1], %o1
tst %o1
bne loop
nop
end:
set fmt, %o0
mov 0x0A, %o1
call printf, 2
nop
ret
restore
```

What gets printed? _______________________ (4 points)

What is the value of each of the following expressions taken sequentially based on changes that may have been made in previous statements? (6 points)

```c
char *a = "BuildOnACommit!";
char *p = a + 7;

* & p --
* -- p + 1
* & (2[ a ])
p[ -2 ]
* a ++
*( a + 3 )
```
<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>
Scratch Paper
Scratch Paper