CSE 30
Spring 2005
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 ___________________ (11 points)
Total ___________________
1. Number Systems

Convert $\text{FB3C}_{16}$ (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 $415_{10}$ to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 points)

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

Convert $-396_{10}$ 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)

<p>| 00101100 | 01110110 | 10101010 |</p>
<table>
<thead>
<tr>
<th>+11010100</th>
<th>+01001101</th>
<th>+11010110</th>
</tr>
</thead>
</table>

| N | Z | V | C |
| N | Z | V | C |
| N | Z | V | C |

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)

```assembly
.globl 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 %l0
  cmp %i0, %i1
  bne L1
  nop
  mov %l0, %i0
  ret
  restore
L2:
  inc %i0
  inc %l0
  cmp %i0, %i1
  bne L2
  nop
  mov %l0, %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)

```asm
.global main  /* main.s */
.section "".text"
main:
    save %sp, -96, %sp
    mov 3, %o0
    mov 2, %o1
    call recurse
    nop
    ret
    restore

.global recurse  /* recurse.s */
.section "".rodata"
fmt: .asciz "%d
"
.section "".text"
recurse:
    save %sp, -(92 + 8) & -8, %sp    ! 2 local int variables
    sub %i0, %i1, %l0
    st %l0, [%fp - 4]
    set fmt, %o0
    mov %l0, %o1
    call printf
    nop
    cmp %i0, 8
    bge L1
    nop
    add %i0, 2, %o0
    mov %i1, %o1
    call recurse
    nop
    ld [%fp - 4], %l0
    add %l0, %o0, %l1
    st %l1, [%fp - 8]
    ba L2
    nop
L1:
    st %l0, [%fp - 8]
L2:
    set fmt, %o0
    ld [%fp - 8], %o1
    call printf
    nop
    ld [%fp - 8], %i0
    ret
    restore
```

Put output here
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. **Draw lines.**

```c
int fubar( char x, short y ) {  // x  y
    int  *local_stack_var1;
    struct foo {  // s1[3] s2 s3[3]
        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)
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;
}
```
8. Floating Point

Convert 126.625_{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 0xC34BC000 (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)

```c
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().

<table>
<thead>
<tr>
<th>Variable</th>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>C</td>
</tr>
<tr>
<td>i</td>
<td>B</td>
</tr>
<tr>
<td>h</td>
<td>C</td>
</tr>
<tr>
<td>g</td>
<td>B</td>
</tr>
<tr>
<td>f</td>
<td>A</td>
</tr>
<tr>
<td>e</td>
<td>C</td>
</tr>
<tr>
<td>d</td>
<td>B</td>
</tr>
<tr>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>b</td>
<td>B</td>
</tr>
<tr>
<td>a</td>
<td>A</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Lifetime</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>B</td>
</tr>
<tr>
<td>i</td>
<td>B</td>
</tr>
<tr>
<td>h</td>
<td>C</td>
</tr>
<tr>
<td>g</td>
<td>B</td>
</tr>
<tr>
<td>f</td>
<td>A</td>
</tr>
<tr>
<td>e</td>
<td>C</td>
</tr>
<tr>
<td>d</td>
<td>B</td>
</tr>
<tr>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>b</td>
<td>B</td>
</tr>
<tr>
<td>a</td>
<td>A</td>
</tr>
</tbody>
</table>
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   0XXXXXXXX

.c:     .byte   0x88

.s:     .half   0xCAFE

.i1:    .word   0x9ABCD123
.i2:    .word   0x9ABCD123
.i3:    .word   0x9ABCD123
.x:     .word   0x00004444

.section ".text"
main:
    save    %sp, -96, %sp

    set     i1, %l0
    set     s, %l1
    ldsh    [%l1], %l1
    sth    %l1, [%l0+2]
    stb    %l1, [%l0+1]

    set     fmt, %o0
    ld    [%l0], %o1
    call     printf ________________________
    nop

    set     i2, %l0
    set     c, %l1
    ldsb    [%l1], %l1
    sth    %l1, [%l0]
    stb    %l1, [%l0+3]

    set     fmt, %o0
    ld    [%l0], %o1
    call     printf ________________________
    nop

    set     x, %l1
    set     i3, %l0
    ldsb    [%l1+2], %l2
    sth    %l2, [%l0]
    stb    %l2, [%l0+2]

    set     fmt, %o0
    ld    [%l0], %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;
}
```

Why do CS majors get Halloween and Christmas mixed up? (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 */
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 (11 points)
Consider the following SPARC assembly program

```assembly
.global main

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

.align 2

section ".text"
main:  
save  %sp, -96, %sp    ! Assume everything above here is the same
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
```

Optimized Version

```assembly
main:
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)

Now optimize the code (in the box above to the right) beginning after the `save` instruction. Everything at and above the `save` instruction cannot be changed. You cannot change the overall algorithm. Some optimizations are worth more than others. Go for the least number of cycles. (7 points)

set   value, regrd  expands to  sethi  %hi(value), regrd
or   regrd, %lo(value), regrd
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 |