CSE 30
Winter 2004
Midterm Exam

1. Number Systems ___________________  (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching and Arrays ___________________  (20 points)
4. Bit Operations / C Runtime Environment ___________________  (17 points)
5. Parameter Passing and Return Values (Structures) ___________________  (12 points)
6. Local Variables, The Stack and Return Values ___________________  (15 points)
7. Load/Store/Memory ___________________  (9 points)

SubTotal ___________________  (100 points)

Extra Credit ___________________  (5 points)

Total ___________________
1. Number Systems

Convert $\text{FB8D}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

- binary: 
- octal: $0$
- decimal: 

Convert $432_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

- sign-magnitude: $0x$
- 1’s complement: $0x$
- 2’s complement: $0x$

Convert $-491_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (6 points)

- sign-magnitude: $0x$
- 1’s complement: $0x$
- 2’s complement: $0x$

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)

<table>
<thead>
<tr>
<th>10111011</th>
<th>+10010100</th>
<th>01010110</th>
<th>+10101010</th>
<th>01111011</th>
<th>+00010100</th>
</tr>
</thead>
<tbody>
<tr>
<td>N Z V C</td>
<td>N Z V C</td>
<td>N Z V C</td>
<td>N Z V C</td>
<td>N Z V C</td>
<td>N Z V C</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Branching and Arrays

Write the SPARC assembly instructions to complete the following. **Do not optimize nops.** (20 points)

```c
int reverseIt( int a[], int low, int high ) {
    int i = 0;
    int tmp = 0;

    while ( (high - low) >= 1 ) {
        ++i;
        tmp = a[low];
        a[low] = a[high];
        a[high] = tmp;
        ++low;
        --high;
    }

    return i;
}
```

```sparc assembly
.global reverseIt
.section ".text"
reverseIt:
    save %sp, -96, %sp
    clr %10
    clr %11

    ! i = 0;
    ! tmp = 0;

    ! %12 <= (high - low)

    ! skip loop if

    ! high-low IS NOT >=1

    loop:
        ++i;

        ! %13 <= low * 4

        ! tmp = a[low];

        ! %15 <= high * 4

        ! %16 <= a[high]

        ! a[low] = a[high];

        ! a[high] = tmp;

        ! ++low;

        ! --high;

        ! %12 <= (high - low)

        ! loop again if

        ! high-low IS >= 1

    end:
        return i;
```
4. Bit Operations / C Runtime Environment

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

```
set 0xC8675309, %l0
sll %l0, 9, %l0

Value in %l0 is 0x_______________________________ (2 points)
```

```
set 0xC8675309, %l0
sra %l0, 10, %l0

Value in %l0 is 0x_______________________________ (2 points)
```

```
set 0xC8675309, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0 ! Value in %l0 is now 0xCAFEBABE

Value set in %l1 must be this bit pattern 0x_______________________________ (3 points)
```

Fill in the names of the 5 areas of the C Runtime Environment as layed out by the SPARC architecture. Then state what parts of a C program are in each area. (10 points)

```
<table>
<thead>
<tr>
<th>Low Memory</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>High Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

low memory

high memory
Write the equivalent unoptimized SPARC assembly language instructions to perform the following C code fragment. You can assume just this one local variable. (12 points)

C

/* Function Prototype */
short foo( char, long, char );

/* ... Other code ... */

/* Assume this local variable fb is declared appropriately and is the only local variable. */

struct fubar {
    char  a[5];
    char  b;
    long  c;
    short d;
    char  e;
    int   f;
    float g;
} fb;   /* Local variable fb */

/* ... Other code ... */

*/

Write the code for just this function call saving the return value appropriately.
*/

fb.d = foo( fb.a[2], fb.c, fb.e );
6. Local Variables, The Stack, and Return Values
Here is a C function that doesn’t do much but allocate local variables, perform statements, and returns a value:

```c
int fubar ( int a, int b ) {
    char  local_stack_var1[8];
    char *local_stack_var2;

    local_stack_var1[3] = a + 'A';               /* statement 1 */
    local_stack_var2 = &local_stack_var1[6];     /* statement 2 */
    --local_stack_var2;                          /* statement 3 */
    *(local_stack_var2 + 2) = '?';               /* statement 4 */
    return ( b + (local_stack_var1[3] - 'a') );  /* statement 5 */
}
```

Now write the equivalent **unoptimized** SPARC assembly language instructions to perform the equivalent. **You must allocate all local variables on the Stack.** Perform each instruction literally. **No short-cuts.** Draw a line between groups of instructions to indicate which instructions are associated with each C statement. (15 points)

**SPARC assembly**
```assembly
.global  fubar
.section  "text"
fubar:  /* Your **unoptimized** code goes below this point */
```
7. Load/Store/Memory
What gets printed in the following program? (9 points)

```
.global main

.section ".data"
fmt: .asciz "0x%x\n" ! prints value as hex 0xXXXXXX

.c:
.byte 0xDD

.s:
.align 2
h: .half 0x89AB

.i1: .word 0xA2345678
i2: .word 0xA2345678
i3: .word 0xA2345678

.x: .word 0

.section ".text"
main:
    save %sp, -96, %sp
    set i1, %l0
    set c, %l1
    ldsb [%l1], %l1
    sth %l1, [%l0]
    set fmt, %o0
    ld [%l0], %o1
    call printf
    nop
    set i2, %l0
    set s, %l1
    ldsh [%l1], %l1
    sth %l1, [%l0+2]
    set fmt, %o0
    ld [%l0], %o1
    call printf
    nop
    set i3, %l0
    set x, %l1
    ldsb [%l0], %l2
    sth %l2, [%l1]
    ldsh [%l0], %l2
    stb %l2, [%l1+3]
    mov %l1, %l0
    set fmt, %o0
    ld [%l0], %o1
    call printf
    nop
    ret
    restore
```
Extra Credit (5 points)
Write the following function to calculate whether a year is a leap year in SPARC Assembly with nops unoptimized on the left side and optimized with no nops on the right side.

```c
int leapyear( unsigned int year ) {
    if ( (year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0)) )
        return 1;
    else
        return 0;
}
```

Unoptimized version here

Optimized version here
Scratch Paper
Scratch Paper