CSE 30
Spring 2006
Midterm Exam

1. Number Systems ___________________ (15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________ (12 points)
3. Branching ___________________ (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 \(0xF913\) (2’s complement, 16-bit word) to the following. (6 points)

- **binary**: _____________________________
- **octal**: 0___________________________
- **decimal**: ___________________________

Convert -361 to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 points)

- **sign-magnitude** \(0x\)___________________________
- **1’s complement** \(0x\)___________________________
- **2’s complement** \(0x\)___________________________

Convert +458 to the following (assume 16-bit word). **Express answers in hexadecimal.** (3 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)

\[
\begin{array}{c}
\begin{array}{c}
11000101 \\
+00111001
\end{array} & \begin{array}{c}
01111111 \\
+00000001
\end{array} & \begin{array}{c}
11010111 \\
+10001001
\end{array}
\end{array}
\]

<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>
</tbody>
</table>

<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>
</tbody>
</table>

<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>
</tbody>
</table>
3. Branching (20 points)
Translate the SPARC Assembly function below into the equivalent C code. Just perform a direct translation. The C function definition is started for you indicating the names of formal parameters and local variables (and their location on the stack). **Do not gotos – use only standard looping and conditional statements.**

**SPARC ASSEMBLY**

```
.section ".text"

bash:           int x;       /* %fp - 4 and %l0 */
                int y;       /* %fp - 8 and %l1 */

            save   %sp, -(92 + 8) & -8, %sp
            st     %i0, [%fp - 4]
            mov    %i0, %l0
            set    8765, %l1
            st     %l1, [%fp - 8]
            cmp    %l1, %i1
            bge    L1
            nop

L2:            sll     %i0, 4, %l1
            st     %l1, [%fp - 8]
            cmp    %l0, %l1
            be     L3
            nop
            add    %l0, %l1, %l0
            st     %l0, [%fp - 4]
            ba     L4
            nop

L3:            sub    %i0, %i1, %l0
            st     %l0, [%fp - 4]
            xor    %l0, 0xBB, %l0
            st     %l0, [%fp - 4]

L4:            cmp    %l1, %i1
            bl     L2
            nop

L1:            ld     [%fp - 8], %i0
            ret
            restore
```

**C**

```
int baz( int a, int b )
{
    int x;       /* %fp - 4 and %l0 */
    int y;       /* %fp - 8 and %l1 */
    save   %sp, -(92 + 8) & -8, %sp
    st     %i0, [%fp - 4]
    mov    %i0, %l0
    set    8765, %l1
    st     %l1, [%fp - 8]
    cmp    %l1, %i1
    bge    L1
    nop

L2:            sll     %i0, 4, %l1
            st     %l1, [%fp - 8]
            cmp    %l0, %l1
            be     L3
            nop
            add    %l0, %l1, %l0
            st     %l0, [%fp - 4]
            ba     L4
            nop

L3:            sub    %i0, %i1, %l0
            st     %l0, [%fp - 4]
            xor    %l0, 0xBB, %l0
            st     %l0, [%fp - 4]

L4:            cmp    %l1, %i1
            bl     L2
            nop

L1:            ld     [%fp - 8], %i0
            ret
            restore
```
4. Bit Operations / C Runtime Environment

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

```plaintext
set  0x9B9C4321, %l0
sra %l0, 11, %l0
Value in %l0 is 0x_______________________________________ (2 points)
```

```plaintext
set  0x9B9C4321, %l0
sll %l0, 13, %l0
Value in %l0 is 0x_______________________________________ (2 points)
```

```plaintext
set  0x9B9C4321, %l0
set  0x????????, %l1
xor %l0, %l1, %l0 ! Value in %l0 is now 0xBABECAFE
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 laid out by the SPARC architecture. Then state what parts of a C program are in each area. (10 points)

```
low memory

__________________________________________________________

__________________________________________________________

__________________________________________________________

__________________________________________________________

```

```
  ↓

high memory
```

---

4
5. Parameter Passing and Return Values (Structures)

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 */

char foo( char, unsigned short, int );

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

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

struct fubar {
    char a;
    unsigned short b[2];
    char c;
    int d[2];
} fb; /* Local variable fb */

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

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

fb.c = foo( fb.a, fb.b[0], fb.d[1] );
```

```sparc
/* Your SPARC assembly code */
```
Here is a C function that doesn’t do much but allocate local variables, perform statements, and returns a value:

```c
int fubar( int x, int y ) {
    int  local_stack_var1[4];
    int *local_stack_var2;

    *local_stack_var2 = 12345;                 /* statement 1 */
    y = *local_stack_var2++;                   /* statement 2 */
    local_stack_var1[0] = x;                   /* statement 3 */
    local_stack_var2 = &local_stack_var1[1];   /* statement 4 */
    return ( y - local_stack_var1[3] );        /* 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)

```
.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%lx\n" ! prints value as hex  0x00000000X
c: .byte 0xDD
.s: .half 0x89AB
.i1: .word 0x87654321
.i2: .word 0x87654321
.i3: .word 0x87654321
.x: .word 0

.section ".text"
main:
  save  %sp, -96, %sp
  set   i1, %l0
  set   s, %l1
  ldshb [%l1], %l1  %l1, [%l1+2]
  sth   %l1, [%l1+2]
  set   fmt, %o0
  ld    [%l0], %ol
  call  printf _________________________________
  nop
  set   i2, %l0
  set   c, %l1
  ldsb  [%l1], %l1
  sth   %l1, [%l1]
  set   fmt, %o0
  ld    [%l0], %ol
  call  printf _________________________________
  nop
  set   i3, %l0
  set   x, %l1
  ldshb [%l1+1], %l2
  sth   %l2, [%l1+2]
  ldshb [%l1+2], %l2
  stb   %l2, [%l1]
  mov   %l1, %l0
  set   fmt, %o0
  ld    [%l0], %ol
  call  printf _________________________________
  nop

ret
restore
Extra Credit (5 points)

Optimize the following SPARC Assembly code fragment. You can assume there are other instructions above and below this code fragment, but only optimize using the instructions given in this code fragment. Some optimizations may be worth more than others.

Unoptimized SPARC Assembly

bash:
    save  $sp, -96, $sp
    mov  $i0, $l0
    set  5678, $l1
    cmp  $l1, $i1
    bge  L1
    nop
    L2:
    sll  $i0, 2, $l1
    cmp  $l0, $l1
    be   L3
    nop
    add  $l0, $l1, $l0
    ba   L4
    nop
    L3:
    sub  $i0, $i1, $l1
    xor  $l0, 0xFF, $l0
    L4:
    cmp  $l1, $i1
    bl   L2
    nop
L1:
    /* Other code you cannot use */
Scratch Paper