CSE 30
Spring 2005
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 \(0xB19\) (2’s complement, 16-bit word) to the following. (6 points)

- **binary**
  ______________________________________
- **octal**
  0___________________________________
- **decimal**
  ______________________________________

Convert +346 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 -455 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)

\[
\begin{array}{c}
10111011 \\
+00010100
\end{array} \quad \begin{array}{c}
11010110 \\
+10111001
\end{array} \quad \begin{array}{c}
01010111 \\
+00101001
\end{array}
\]

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

```c
int initArray( int *begin, int *stop, int val, int inc )
{
    int howMany = 0;

    while ( begin != stop )
    {
        *begin++ = val;     /* *begin = val; begin++; */
        val = val + inc;
        howMany++;
    }

    return howMany;
}
```
What is the value of `%l0` after each statement is executed? **Express your answers in hexadecimal.**

```c
set 0x9876ACED, %l0
sll %l0, 13, %l0
```

Value in `%l0` is `0x` ____________________________ (2 points)

```c
set 0x9876ACED, %l0
sra %l0, 9, %l0
```

Value in `%l0` is `0x` ____________________________ (2 points)

```c
set 0x9876ACED, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0 ! Value in %l0 is now 0xDEADBEEF
```

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)

<table>
<thead>
<tr>
<th>low memory</th>
<th>high memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(10 points)
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
struct fubar {
    char           a;
    int            b;
    short          c[3];
    char           d[3];
    unsigned short e;
} fb;   /* Local variable fb */

fb.c[2] = foo( fb.d[1], fb.e, fb.a );
```

---

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 */
short foo( char, unsigned short, char );

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

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

struct fubar {
    char           a;
    int            b;
    short          c[3];
    char           d[3];
    unsigned short e;
} fb;   /* Local variable fb */

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

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

fb.c[2] = foo( fb.d[1], fb.e, fb.a );
```
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( short a, char b ) {
    int *local_stack_var1;
    int local_stack_var2[5];

    local_stack_var2[3] = a + 321;               /* statement 1 */
    local_stack_var1 = &local_stack_var2[3];     /* statement 2 */
    local_stack_var1++;                          /* statement 3 */
    *(local_stack_var1 + 1) = 123;               /* statement 4 */
    return ( b + (local_stack_var2[4] - 420) );  /* 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
.globa1 fubar
.sectio1n "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 0x00000000

c: .byte 0x44

.align 2
s: .half 0x5432

.align 4
i1: .word 0xACED9876
i2: .word 0xACED9876
i3: .word 0xACED9876
x: .word 0

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

    set i1, %l0
    ldsb [%l1+1], %l1
    sth %l1, [%l0+2]

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

set i2, %l0

set c, %l1
ldsb [%l1+1], %l1
stb %l1, [%l0+1]

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

set i3, %l0

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

mov %l1, %l0

set fmt, %o0
ld [%l0], %o1
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.

<table>
<thead>
<tr>
<th>Unoptimized SPARC Assembly</th>
<th>Optimized SPARC Assembly</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Other code you cannot use */</td>
<td></td>
</tr>
<tr>
<td>cmp %i1, %l3</td>
<td></td>
</tr>
<tr>
<td>bg end_loop</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td></td>
</tr>
<tr>
<td>add %i1, %l5, %i1</td>
<td></td>
</tr>
<tr>
<td>mov %i1, %o0</td>
<td></td>
</tr>
<tr>
<td>mov 64, %o1</td>
<td></td>
</tr>
<tr>
<td>call .mul</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>mov %o0, %i1</td>
<td></td>
</tr>
<tr>
<td>cmp %i1, %l3</td>
<td></td>
</tr>
<tr>
<td>ble loop</td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
</tr>
<tr>
<td>end_loop:</td>
<td></td>
</tr>
<tr>
<td>inc %i1</td>
<td></td>
</tr>
<tr>
<td>/* Other code you cannot use */</td>
<td></td>
</tr>
</tbody>
</table>
Scratch Paper