CSE 30  
Winter 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 0xFB7A (2’s complement, 16-bit word) to the following. (6 points)

binary ____________________________________________________
octal 0 __________________________________________
decimal ____________________________________________________

Convert -393 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 +506 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)

\[
\begin{array}{ccc}
01010111 & +00101001 & 10111011 \\
11010110 & +00111001 & +10010100 \\
\end{array}
\]

\[
\begin{array}{cccc}
N & Z & V & C \\
\hline
| & | & | & | \\
| & | & | & | \\
| & | & | & | \\
\end{array}
\]
3. Branching (20 points)
Write the SPARC assembly instructions to complete the following. **Do not optimize nops.** (20 points)

```c
/* Return index of n in b; -1 if not found. */
int findIt( int n, int b[], int size ) {
    int i;
    .global  findIt
    .section  ".text"
    findIt:
    for ( i = 0; i < size; ++i ) {
        if ( *b == n )
            return i;
        ++b;
    }
    return -1;
}
```

```sparc
/* Return index of n in b; -1 if not found. */
int findIt( int n, int b[], int size ) {
    int i;
    .global  findIt
    .section  ".text"
    findIt:
    for ( i = 0; i < size; ++i ) {
        if ( *b == n )
            return i;
        ++b;
    }
    return -1;
}
```
4. Bit Operations / C Runtime Environment

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

```c
set 0xABCDEF19, %l0
sra %l0, 9, %l0
```

Value in %l0 is 0x_______________________________________ (2 points)

```c
set 0xABCDEF19, %l0
sll %l0, 13, %l0
```

Value in %l0 is 0x_______________________________________ (2 points)

```c
set 0xABCDEF19, %l0
set 0x?????????, %l1
xor %l0, %l1, %l0
```

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

Value in %l0 is now 0xFEEDCAFE

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></th>
<th></th>
<th></th>
<th>high memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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( long, short, char );

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

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

struct fubar {
    char  a[3];
    short b;
    long  c;
    short d;
    char  e;
    short f[2];
} fb;   /* Local variable fb */

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

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

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

SPARC assembly
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( char a, int b ) {
    short *local_stack_var1;
    short local_stack_var2[5];

    local_stack_var2[3] = a + 333; /* statement 1 */
    local_stack_var1 = &local_stack_var2[3]; /* statement 2 */
    local_stack_var1++; /* statement 3 */
    *(local_stack_var1 + 1) = 111; /* statement 4 */
    return ( b + (local_stack_var2[4] - 777) ); /* 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
.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)

```assembly
.global main

.section ".data"
fmt: .asciz "0x%x\n" ! prints value as hex 0x00000000X
c: .byte 0xBB
.align 2
s: .half 0x87B9
.align 4
i1: .word 0xABC0D5432
i2: .word 0xABC0D5432
i3: .word 0xABC0D5432
x: .word 0

.section ".text"
main:
  save  %sp, -96, %sp
  set   i1, %l0
  set   s, %l1
  ldsb  [%l1+1], %l1
  sth  %l1, [%l0+2]
  set   fmt, %o0
  ld   [%l0], %o1
  call  printf
  nop

  set   i2, %l0
  set   c, %l1
  ldub  [%l1], %l1
  stb  %l1, [%l0+3]
  set   fmt, %o0
  ld   [%l0], %o1
  call  printf
  nop

  set   i3, %l0
  set   x, %l1
  ldub  [%l0+1], %l2
  stb  %l2, [%l1]
  ldsb  [%l0], %l2
  sth  %l2, [%l1+2]
  mov  %l1, %l0
  set   fmt, %o0
  ld   [%l0], %o1
  call  printf
  nop
ret
restore
```
**Extra Credit (5 points)**
Write a function in SPARC Assembly that takes two unsigned ints and interleaves the lower two bytes into a full-length unsigned integer and returns that interleaved value. For example,

```c
word1: 0x 00 00 CA 73
word2: 0x 00 00 D4 B2
```

interleaves the lower two bytes to produce

```c
result: 0x CA D4 73 B2
```

You may use only the following assembly instructions: `and`, `or`, `sll`, `set` (along with `save`, `ret`, `restore`). (You should be able to do this with 4 `and`, 3 `or`, 3 `sll`, and 1 `set` instructions.)

Here is the function prototype for this function:

```c
unsigned int interleave_bytes(unsigned int word1, unsigned int word2);
```
Scratch Paper