CSE 30  
Fall 2008  
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 ___________________  (11 points)

SubTotal ___________________  (102 points)

Extra Credit ___________________  (5 points)

Total ___________________
1. Number Systems

Convert 0xFA6D (2’s complement, 16-bit word) to the following. (6 points)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>octal</td>
<td>decimal</td>
</tr>
<tr>
<td>________</td>
<td>___________</td>
<td>____________________________</td>
</tr>
<tr>
<td>________</td>
<td>___________</td>
<td>____________________________</td>
</tr>
<tr>
<td>________</td>
<td>___________</td>
<td>____________________________</td>
</tr>
</tbody>
</table>

Convert -462 to the following (assume 16-bit word). Express answers in hexadecimal. (6 points)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sign-magnitude</td>
<td>1’s complement</td>
<td>2’s complement</td>
</tr>
<tr>
<td>0x_________</td>
<td>___________</td>
<td>___________</td>
</tr>
</tbody>
</table>

Convert +325 to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sign-magnitude</td>
<td>1’s complement</td>
<td>2’s complement</td>
</tr>
<tr>
<td>0x_________</td>
<td>___________</td>
<td>___________</td>
</tr>
</tbody>
</table>

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>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11000101</td>
<td>01111111</td>
<td>11010111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+00111001</td>
<td>+00000001</td>
<td>+10001001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td>N</td>
<td>Z</td>
<td>V</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
<td>__________</td>
<td>__________</td>
<td>__________</td>
<td>__________</td>
<td>__________</td>
<td></td>
<td></td>
<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 use gotos/breaks/continues – just standard looping/conditional stmts.**

```
SPARC ASSEMBLY

.section ".text"

baz:
    save  %sp, -(92 + 8) & -8, %sp
    st    %i0, [%fp - 4]
    mov   %i0, %l0
    set   5678, %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, %l1, %l0
    st    %l0, [%fp - 4]
    xor   %l0, 0xDD, %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 %10 */
    int y; /* %fp - 8 and %11 */
...
4. Bit Operations / C Runtime Environment

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

```
set  0xABCD9753, %l0
sra  %l0, 11, %l0
```

Value in %l0 is **0x_________________________** (2 points)

```
set  0xABCD9753, %l0
sll  %l0, 13, %l0
```

Value in %l0 is **0x_________________________** (2 points)

```
set  0xABCD9753, %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)

<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>
</tbody>
</table>

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] );
```

Put your SPARC Assembly code in the box below.
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

```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)

SPARC assembly

```
.global fubar
.section " .text"

fubar:  /* Your unoptimized code goes below this point */
```
11. Load/Store/Memory
Specify the hex values requested after those lines have been fully executed. (11 points)

```assembly
.global main
.section ".data"
fmt: .asciz "0x%08X\n" ! prints value as hex 0XXXXXXXX
.c: .byte 0xDD
.s: .half 0xBABE
    .align 4
.i1: .word 0x12345678
.i2: .word 0x12345678
.i3: .word 0x12345678
.x: .word 0x77770000
    .section ".text"
main:
    save %sp, -96, %sp
    set x, %l0
    set s, %l1
  ldsh [%l1], %l2                      Hex value in %l2
  ld  [%l1], %l2                      Hex value in word labeled x
  stb  %l2, [%l0+1]                  Hex value in %l2
  srl  %l2, 12, %l2                  Hex value in %l2
  stb  %l2, [%l0+3]
    set fmt, %o0
    ld  [%l0], %o1
    call printf                        Hex value in word labeled x
    nop (same as output of this printf)
    set i1, %l0
    set c, %l1
  ldub [%l1], %l2                     Hex value in %l2
  sth  %l2, [%l0+2]                  Hex value in word labeled i1
  stb  %l2, [%l0]
    set fmt, %o0
    ld  [%l0], %o1
    call printf                        Hex value in word labeled i1
    nop (same as output of this printf)
    set i2, %l0
    set i3, %l1
  ld  [%l1], %l2                     Hex value in %l2
  sth  %l2, [%l0]                    Hex value in word labeled i2
  sra  %l2, 16, %l2                  Hex value in %l2
  stb  %l2, [%l0+1]
    set fmt, %o0
    ld  [%l0], %o1
    call printf                        Hex value in word labeled i2
    nop (same as output of this printf)
    ret
    restore
```
**Extra Credit (5 points)**

Optimize the following SPARC Assembly code fragment by filling the nops in the delay slots. 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**

/* Other code you cannot use */

`baz:
  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, %l0
  xor %l0, 0xFF, %l0
L4:
  cmp %l1, %i1
  bl L2
  nop
L1:
/* Other code you cannot use */`

**Optimized SPARC Assembly**

/* Other code you cannot use */

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