

Name _____

Signature _____

cs30x _____

Student ID _____

**CSE 30
Fall 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 **0xF939** (2's complement, 16-bit word) to the following. (6 points)

binary _____

octal **0** _____

decimal _____

Convert **-328** 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 **+477** 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)

$$\begin{array}{r} 11010111 \\ +10001001 \\ \hline \end{array}$$

$$\begin{array}{r} 11000101 \\ +00111001 \\ \hline \end{array}$$

$$\begin{array}{r} 00101111 \\ +01010001 \\ \hline \end{array}$$

N Z V C

 | | | |

N Z V C

 | | | |

N Z V C

 | | | |

3. Branching (20 points)

Translate the C function below into the equivalent SPARC Assembly code. Just perform a direct translation – no optimizations. The assembly function is started for you. **Do not gotos – only use standard looping and conditional statements.**

C

```
int fubar( int uno, int duo )
{
    if ( uno > duo )
    {
        while ( duo < 57 )
        {
            uno = duo - 10;
            ++duo;
        }
    }
    else
    {
        duo = uno + duo;
    }
    return duo - uno;
}
```

SPARC ASSEMBLY

```
fubar:
    .global fubar
    .section ".text"
    save %sp, -96, %sp
```

4. Bit Operations / C Runtime Environment

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

```
set 0x9D9E8765, %l0
sra %l0, 9, %l0
```

Value in %l0 is **0x**_____ (2 points)

```
set 0x9D9E8765, %l0
sll %l0, 14, %l0
```

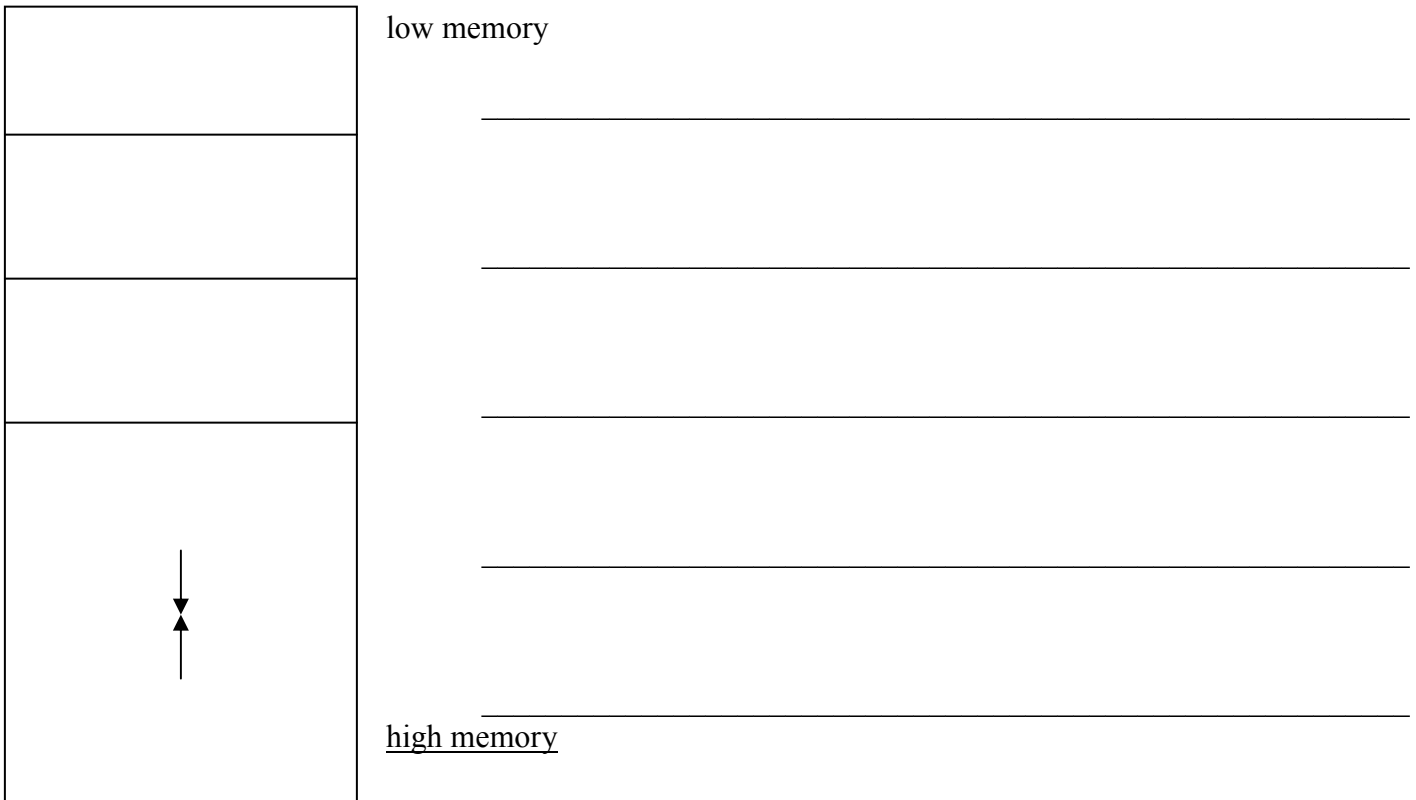
Value in %l0 is **0x**_____ (2 points)

```
set 0x9D9E8765, %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 laid out by the SPARC architecture. Then state what parts of a C program are in each area. (10 points)



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

SPARC assembly

```
/* Function Prototype */

int foo( unsigned short, int, char );

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

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

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

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

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

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

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
int fubar( int x, int y ) {
    long *local_stack_var1;
    long local_stack_var2[3];

    y = local_stack_var2[1];                /* statement 1 */
    local_stack_var1 = local_stack_var2 + 2; /* statement 2 */
    *local_stack_var1 = 420024;             /* statement 3 */
    --local_stack_var1;                     /* statement 4 */
    return ( x + local_stack_var2[2] );     /* 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 */
```

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 0XXXXXXXXX

c:   .byte 0xEE

    .align 2
s:   .half 0x8ABC

    .align 4
i1:  .word 0x24681357
i2:  .word 0x24681357
i3:  .word 0x24681357
x:   .word 0

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

    set     i1, %10

    set     c, %11
    ldsb   [%11], %11
    sth    %11, [%10]

    set     fmt, %0
    ld     [%10], %01
    call   printf
    nop

    set     i2, %10

    set     x, %11
    ldub   [%10+1], %12
    sth    %12, [%11+2]
    ldsh   [%10+2], %12
    stb    %12, [%11]

    mov    %11, %10

    set     fmt, %0
    ld     [%10], %01
    call   printf
    nop

    set     i3, %10

    set     s, %11
    ldub   [%11], %12
    stb    %12, [%10+1]
    ldub   [%11+1], %12
    stb    %12, [%10+2]

    set     fmt, %0
    ld     [%10], %01
    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

```
/* 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, 3, %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

Scratch Paper

Scratch Paper