

Name _____

Signature _____

cs30x _____

Student ID _____

**CSE 30
Fall 2007
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 (Stack Variables) _____ **(12 points)**

6. Local Variables, The Stack and Return Values _____ **(15 points)**

7. Load/Store/Memory _____ **(9 points)**

SubTotal _____ **(100 points)**

Extra Credit _____ **(6 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)

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

N Z V C

| | | | |

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

N Z V C

| | | | |

$$\begin{array}{r} 01111111 \\ +00000001 \\ \hline \end{array}$$

N Z V C

| | | | |

3. Branching (20 points)

Translate the C code below into the equivalent **unoptimized** SPARC Assembly code. Just perform a direct translation – no optimizations. Use the local register mappings for the variables in assembly as specified.

C

```
/* Assume variables x and y have been
   properly declared as ints. */

for ( x = 5678; x >= 420; x = x - 70 )
{
    y = x;

    if ( y < 747 )
    {
        y = x - 15;
    } else {
        x = y + 95;
    }
}
```

SPARC ASSEMBLY

```
! x is mapped to %11
! y is mapped to %14
```

4. Bit Operations / C Runtime Environment

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

```
set 0x9B9C4321, %l0
sra %l0, 13, %l0
```

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

```
set 0x9B9C4321, %l0
sll %l0, 11, %l0
```

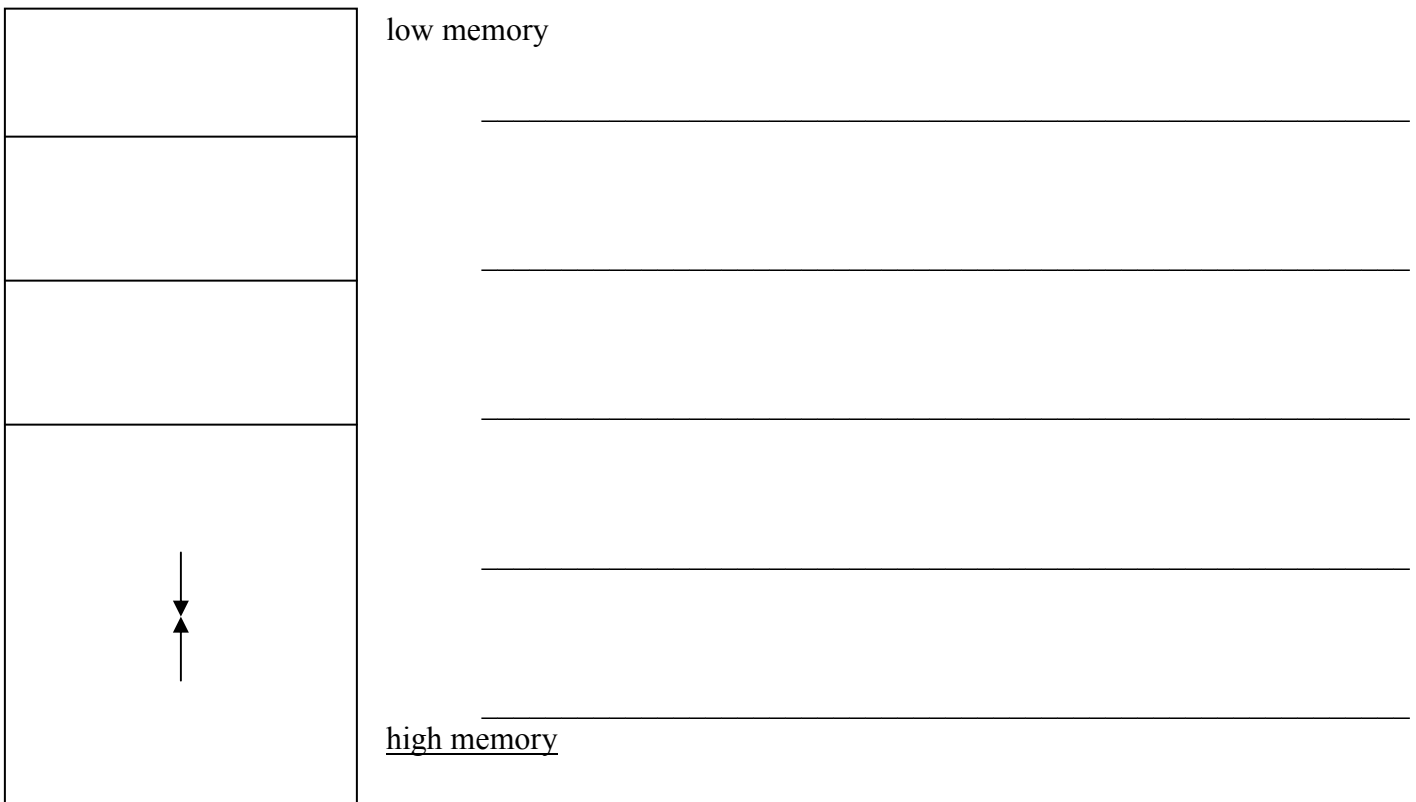
Value in %l0 is **0x**_____ (2 points)

```
set 0x9B9C4321, %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 (Local Stack Variables)

Write the equivalent **unoptimized** SPARC assembly language instructions to perform the following C code fragment. **All local variables must be allocated on the run time stack.** (12 points)

C

```
/* Function Prototype */

char foo( unsigned short, int, char );

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

/* Local stack variables */

char          a;
unsigned short b[2];
char          c;
int           d[2];

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

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

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

SPARC assembly

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 ) {
    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 */
```

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 0xBB

    .align 2
s:   .half 0x9876

    .align 4
i1:  .word 0x12345678
i2:  .word 0x12345678
i3:  .word 0x12345678
x:   .word 0

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

    set     i2, %10

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

    set     fmt, %0
    ld     [%10], %o1
    call   printf
    nop

    set     i3, %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], %o1
    call   printf
    nop

    set     i1, %10

    set     s, %11
    ldub   [%11], %11
    sth    %11, [%10+2]

    set     fmt, %0
    ld     [%10], %o1
    call   printf
    nop

    ret
    restore
```

Extra Credit (6 points)

What gets printed at each printf() statement given the following C program?

```
#include <stdio.h>

int
main()
{
    char s[] = "absolute";
    char *p = s;

    printf( "%c\n", *p++ );           _____
    --*(p+4);
    printf( "%c\n", *++p );         _____
    p = p+1;
    *p = *(p-3) + 4;
    printf( "%c\n", p[0] );         _____
    *(p+1) = p[1] + 2;
    printf( "%c\n", *++p );         _____
    p++;
    printf( "%c\n", *p++ );         _____
    p[0] = *(p+1);
    printf( "%s\n", s );           _____

    return 0;
}
```

Scratch Paper

Scratch Paper