

Name _____

Signature _____

cs30x _____

Student ID _____

**CSE 30
Fall 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 **0xFA21** (2's complement, 16-bit word) to the following. (6 points)

binary _____

octal **0** _____

decimal _____

Convert **-326** 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 **+435** 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)

```
  01010111
+ 00101001
-----
```

N Z V C

```
-----
|   |   |   |   |
-----
```

```
  11010110
+10111001
-----
```

N Z V C

```
-----
|   |   |   |   |
-----
```

```
  10111011
+00010100
-----
```

N Z V C

```
-----
|   |   |   |   |
-----
```

3. Branching (20 points)

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

C

SPARC assembly

```
int
fubar( int value, int decAmt, int *head, int *tail )
{
    int count = 0;

    while ( head != tail )
    {
        count++;
        *head++ = value;      /* *head = value; head++; */
        value = value - decAmt;
    }

    return count;
}
```

4. Bit Operations / C Runtime Environment

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

```
set 0x98ACED76, %l0
sra %l0, 13, %l0
```

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

```
set 0x98ACED76, %l0
sll %l0, 11, %l0
```

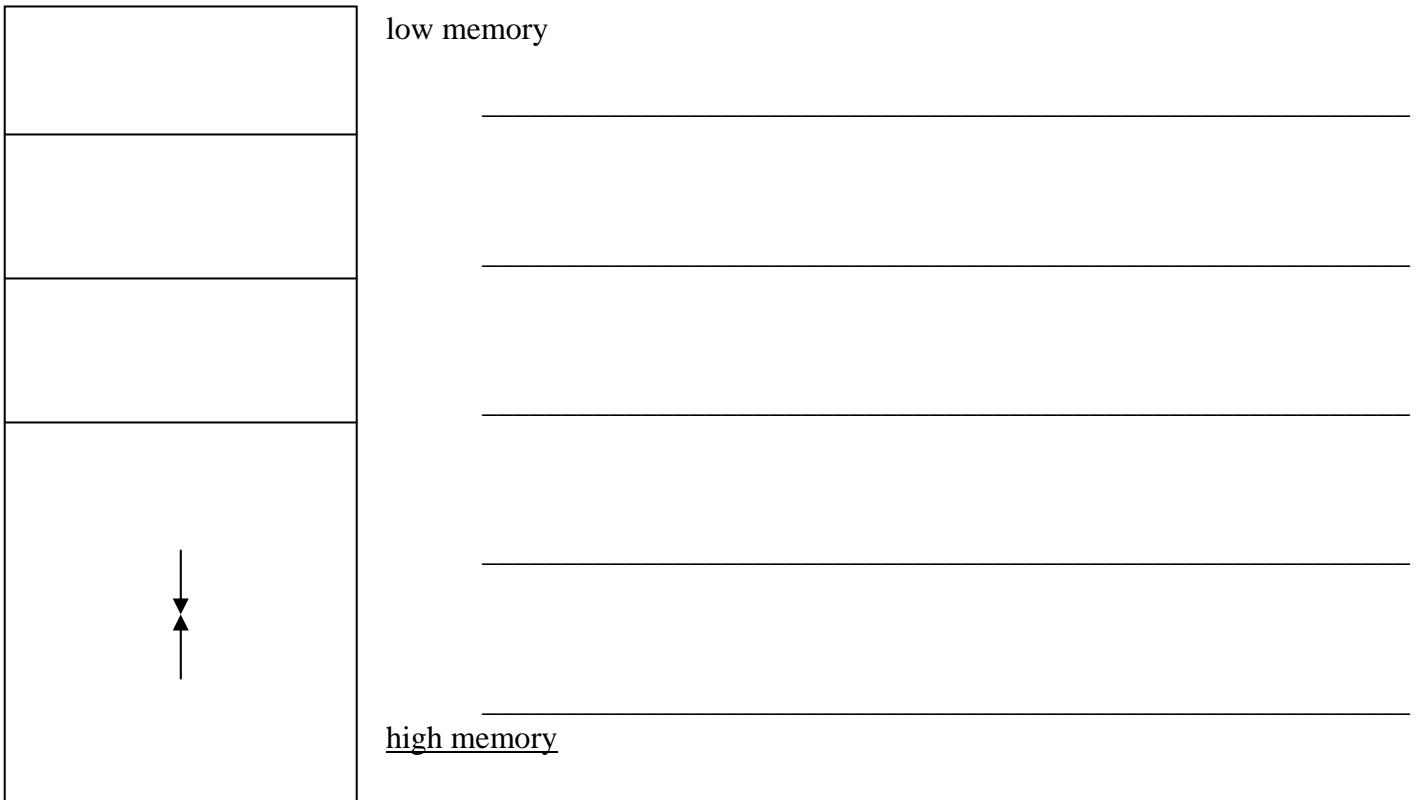
Value in %l0 is **0x**_____ (2 points)

```
set 0x98ACED76, %l0
set 0x????????, %l1
xor %l0, %l1, %l0
```

! Value in %l0 is now OxCAFEBABE

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 */

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

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[4];

    local_stack_var2[3] = b - 123;          /* statement 1 */
    local_stack_var1 = &local_stack_var2[2]; /* statement 2 */
    local_stack_var1++;                    /* statement 3 */
    *(local_stack_var1 + 2) = 321;         /* statement 4 */
    return ( a + (local_stack_var2[1] + 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 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 0xDD

    .align 2
s:   .half 0x2345

    .align 4
i1:  .word 0x9876ACED
i2:  .word 0x9876ACED
i3:  .word 0x9876ACED
x:   .word 0

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

    set     i1, %10

    set     s, %11
    ldsb   [%11+1], %11
    sth    %11, [%10+2]

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

    set     i2, %10

    set     c, %11
    ldub   [%11], %11
    stb    %11, [%10+1]

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

    set     i3, %10

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

    mov    %11, %10

    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 */

    cmp    %i2, %l5
    bg    end_loop
    nop

loop:
    add    %i2, %i0, %i2

    mov    32, %o0
    mov    %i2, %o1
    call   .mul
    nop

    mov    %o0, %i2

    cmp    %i2, %l5
    ble   loop
    nop

end_loop:
    inc    %i2

/* Other code you cannot use */
```

Optimized SPARC Assembly

Scratch Paper

Scratch Paper