

Name \_\_\_\_\_

Signature \_\_\_\_\_

cs30x \_\_\_\_\_

Student ID \_\_\_\_\_

**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)

$$\begin{array}{r} 01010111 \\ +00101001 \\ \hline \end{array}$$

**N    Z    V    C**

-----  
 |   |   |   |   |  
 -----

$$\begin{array}{r} 11010110 \\ +00111001 \\ \hline \end{array}$$

**N    Z    V    C**

-----  
 |   |   |   |   |  
 -----

$$\begin{array}{r} 10111011 \\ +10010100 \\ \hline \end{array}$$

**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

```
/* Return index of n in b; -1 if not found. */
int findIt( int n, int b[], int size ) {
    int i;

    for ( i = 0; i < size; ++i ) {
        if ( *b == n )
            return i;
        ++b;
    }

    return -1;
}

                                .global findIt
                                .section ".text"
                                findIt:
                                save    %sp, -96, %sp
                                /* Complete the rest of this function starting here */
                                ! i mapped to %l4
```

#### 4. Bit Operations / C Runtime Environment

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

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

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

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

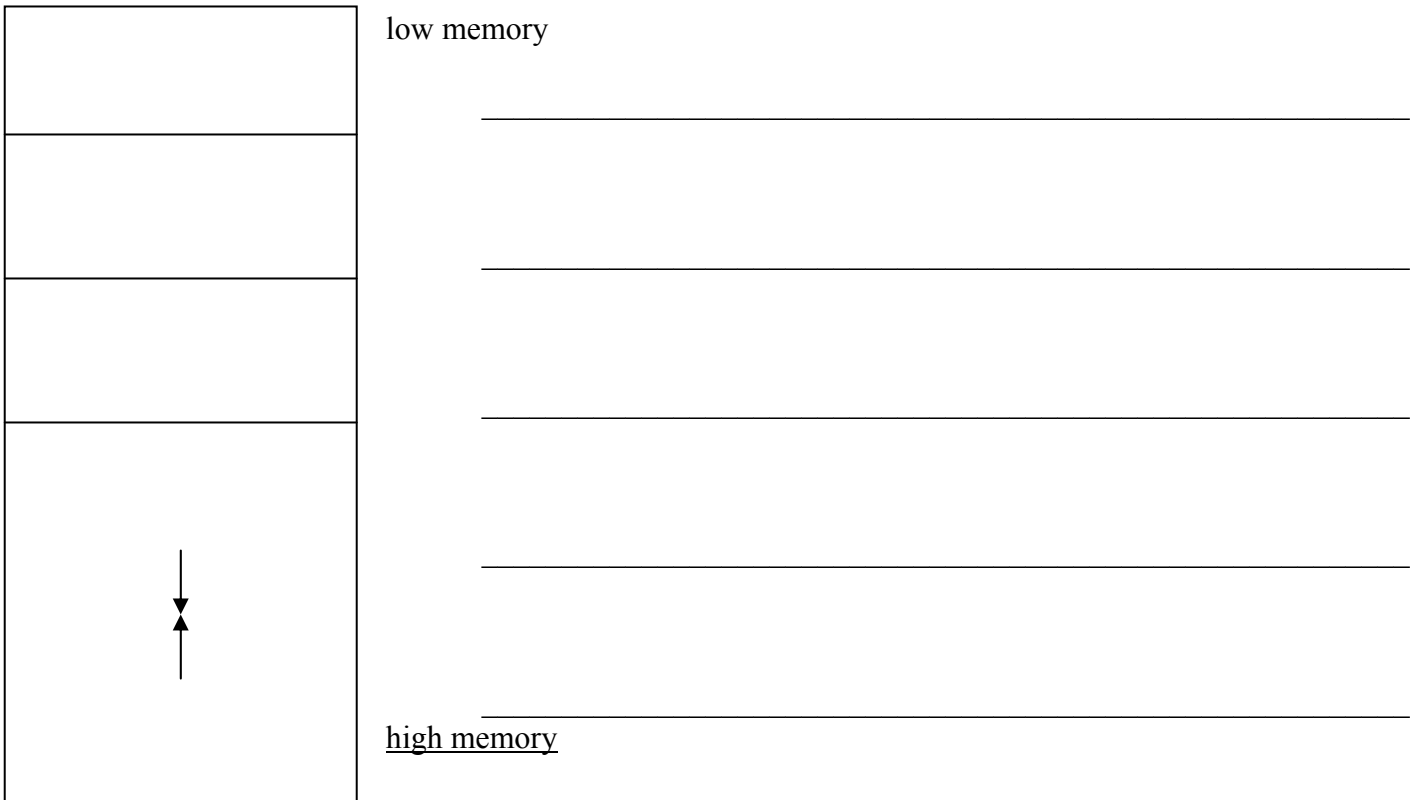
Value in %l0 is **0x**\_\_\_\_\_ (2 points)

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

**! Value in %l0 is now 0xFEEDCAFE**

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 */

char foo( long, short, char );

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

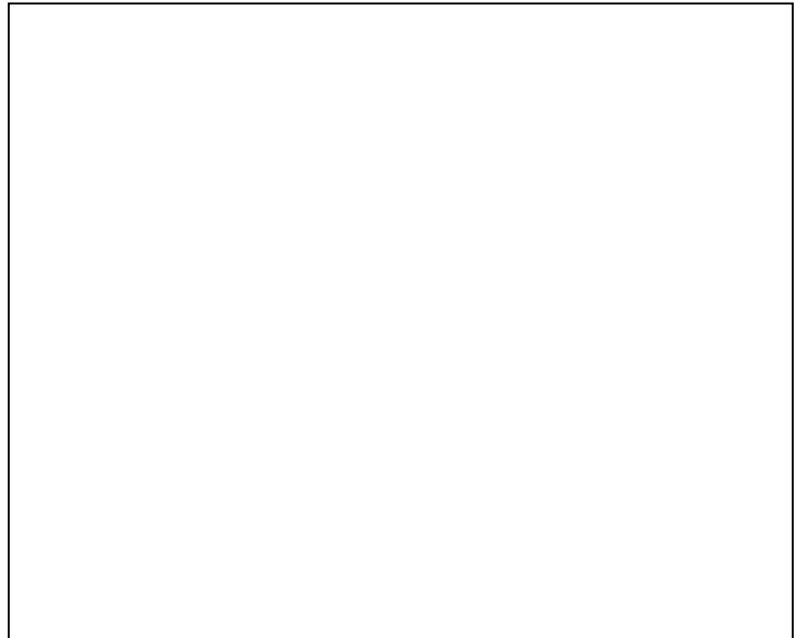
/* Assume this local variable
   is declared appropriately
   and is the only local var. */

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] );
```



## 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 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 0x87B9

    .align 4
i1:  .word 0xABCD5432
i2:  .word 0xABCD5432
i3:  .word 0xABCD5432
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+3]

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

    set     i3, %10

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

    mov    %11, %10

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

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

interleaves the lower two bytes to produce

```
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:

```
unsigned int  interleave_bytes( unsigned int word1, unsigned int word2 );
```

## Scratch Paper

## Scratch Paper