

Name \_\_\_\_\_

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

Signature \_\_\_\_\_

cs30x \_\_\_\_\_

**CSE 30  
Winter 2006  
Midterm Exam**

<b>1. Number Systems</b>	_____	<b>(15 points)</b>
<b>2. Binary Addition/Condition Code Bits/Overflow Detection</b>	_____	<b>(12 points)</b>
<b>3. Branching</b>	_____	<b>(20 points)</b>
<b>4. Bit Operations / C Runtime Environment</b>	_____	<b>(17 points)</b>
<b>5. Parameter Passing and Return Values (Structures)</b>	_____	<b>(12 points)</b>
<b>6. Local Variables, The Stack and Return Values</b>	_____	<b>(15 points)</b>
<b>7. Load/Store/Memory</b>	_____	<b>(9 points)</b>
<b>SubTotal</b>	_____	<b>(100 points)</b>
<b>Extra Credit</b>	_____	<b>(5 points)</b>
<b>Total</b>	_____	

## 1. Number Systems

Convert **0xF933** (2's complement, 16-bit word) to the following. (6 points)

**binary** \_\_\_\_\_

**octal**        **0**\_\_\_\_\_

**decimal** \_\_\_\_\_

Convert **-316** 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 **+485** 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)

$$\begin{array}{r} 01010111 \\ +00001001 \\ \hline \end{array}$$

**N    Z    V    C**

--	--	--	--

$$\begin{array}{r} 10000101 \\ +10111001 \\ \hline \end{array}$$

**N    Z    V    C**

--	--	--	--

$$\begin{array}{r} 11111111 \\ +00000001 \\ \hline \end{array}$$

**N    Z    V    C**

--	--	--	--

### 3. Branching (20 points)

Translate the entire function below into the equivalent SPARC assembly code. Just perform a direct translation. Map the local variable to local register %l3. Do not optimize nops or the algorithm. (20 points)

C

SPARC assembly

```
int
fubaz(int location, int *ptr, char ch )
{
    int i;

    for ( i = location; i >= 0; --i )
    {
        *ptr++ = ch;    /* *ptr = ch; ptr++; */
        ++ch;          /* inc ASCII value of ch */
    }

    return *ptr;
}
```

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

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

```
set 0x8A8B1112, %l0  
sra %l0, 13, %l0
```

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

```
set 0x8A8B1112, %l0  
sll %l0, 11, %l0
```

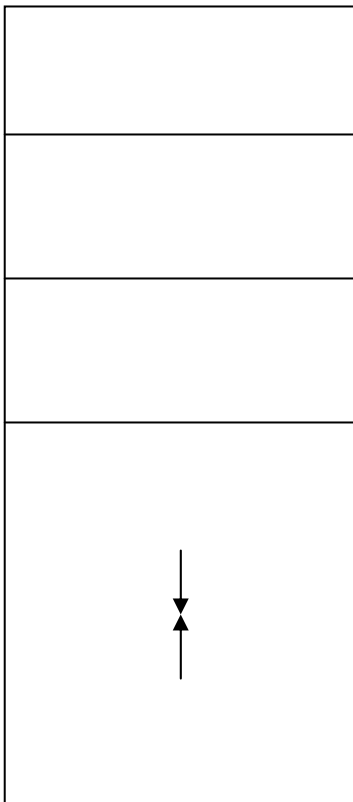
Value in %l0 is **0x**\_\_\_\_\_ (2 points)

```
set 0x8A8B1112, %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)



low memory

---

---

---

---

---

---

---

high memory

## 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, short, char );

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

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

struct fubar {
    char          a[5];
    unsigned short b;
    int           c;
    short         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.b, fb.d[1], fb.a[3] );
```

## 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 x, short y ) {
    short  local_stack_var1[3];
    short *local_stack_var2;

    *local_stack_var2 = 420024;          /* statement 1 */
    --local_stack_var2;                 /* statement 2 */
    y = local_stack_var1[1];            /* statement 3 */
    local_stack_var2 = local_stack_var1 + 2; /* statement 4 */
    return ( x + local_stack_var1[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 0x88

    .align 2
s:   .half 0xBA89

    .align 4
i1:  .word 0xACDC1234
i2:  .word 0xACDC1234
i3:  .word 0xACDC1234
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], %01
    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], %01
    call   printf
    nop

    set     i1, %10

    set     s, %11
    ldub   [%11], %11
    sth    %11, [%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 */

    cmp    %i2, %l5
    bg    L1
    nop

    add    %i2, %i0, %i2

    mov    256, %o0
    mov    %i2, %o1
    call   .mul
    nop

    mov    %o0, %i2

    ba    L2
    nop

L1:
    add    %l5, %l1, %l5
    sub    %i2, %i0, %i2

L2:
    inc    %i2

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

#### Optimized SPARC Assembly

## Scratch Paper

## Scratch Paper