

Name _____

Signature _____

cs30x _____

Student ID _____

**CSE 30
Spring 2007
Midterm Exam**

1. Number Systems	_____	(12 points)
2. Binary Addition/Condition Code Bits/Overflow Detection	_____	(8 points)
3. Branching	_____	(10 points)
4. Bit Operations / C Runtime Environment	_____	(7 points)
5. Parameter Passing and Return Values (Structures)	_____	(12 points)
6. Local Variables, The Stack and Return Values	_____	(12 points)
7. Load/Store/Memory	_____	(3 points)
SubTotal	_____	(64 points)
Extra Credit	_____	(3 points)
Total	_____	

1. Number Systems

Convert **0xFB1D** (2's complement, 16-bit word) to the following. (6 points)

binary _____ (just a direct base translation)
octal **0** _____ (just a direct base translation)
decimal _____ (convert to signed decimal value)

Convert **-428** to the following (assume 16-bit word). **Express answers in hexadecimal.** (6 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. (8 points)

```

  01111111
+10000001
-----

```

```

  01001101
+00111010
-----

```

N	Z	V	C

N	Z	V	C

3. Branching (10 points)

Translate the SPARC Assembly function below into the equivalent C code. Just perform a direct translation. The C function definition is started for you indicating the names of formal parameters and local variables (and their location on the stack). **Do not gotos – use only standard looping and conditional statements.**

<u>SPARC ASSEMBLY</u>	<u>C</u>
<pre>.section ".text" baz: save %sp, -96, %sp cmp %i1, %i0 bg L2 nop L1: cmp %i0, 200 bl L3 nop add %i1, 15, %i1 L3: cmp %i1, %i0 ble L1 nop L2: sub %i0, %i1, %i0 ret restore</pre>	<pre>int baz(int a, int b) { } }</pre>

4. Bit Operations / C Runtime Environment

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

```
set 0xADBC9562, %l0
sra %l0, 11, %l0
```

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

```
set 0xADBC9562, %l0
sll %l0, 9, %l0
```

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

```
set 0xADBC9562, %l0
set 0x????????, %l1
xor %l0, %l1, %l0
```

! Value in %l0 is now 0x87654321

Value set in %l1 must be this bit pattern **0x**_____ (3 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

```
/* Function Prototype */

char foo( int, char, unsigned short );

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

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

struct fubar {
    char          a[3];
    unsigned short b;
    char          c;
    int           d;
} fb; /* Local variable fb */

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

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

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

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[2];
    int *local_stack_var2;

    y = x - local_stack_var1[0];          /* statement 1 */
    local_stack_var1[1] = y % local_stack_var1[0]; /* statement 2 */
    return ( *local_stack_var2++ );      /* statement 3 */
}
```

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. (12 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? (3 points)

```
.global main

.section ".data"
fmt: .asciz "0x%X\n"      ! prints value as hex  0XXXXXXXX

c:   .byte 0xBB

    .align 2
s:   .half 0x89AB

    .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]

    ldsb   [%10+3], %11
    stb   %11, [%10+2]

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

    ret
    restore
```

Extra Credit (3 points)

What gets printed by the following C program?

```
#include <stdio.h>

int
main()
{
    char a[] = "Me? I want to go";
    char b[] = "to Porter's Pub";
    char c[] = "and don't you, too?";
    char *ptr = b;

    printf( "%c\n", *(ptr = ptr + 8) + 1 );
    printf( "%c\n", ptr[0] );
    printf( "%c\n", *(a + 7) );
    printf( "%c\n", b[strlen(a) - 2] );
    printf( "%c\n", toupper( *c + 1 ) );
    printf( "%c\n", ptr[4] );

    return 0;
}
```

Scratch Paper

Scratch Paper