

Name _____

Signature _____

cs30x _____

Student ID _____

**CSE 30
Winter 2009
Midterm Exam**

1. Number Systems	_____	(15 points)
2. Binary Addition/Condition Code Bits/Overflow Detection	_____	(12 points)
3. Branching	_____	(23 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	_____	(16 points)
7. Load/Store/Memory	_____	(11 points)
SubTotal	_____	(106 points)
Extra Credit	_____	(7 points)
Total	_____	

1. Number Systems

Convert **0xFC65** (2's complement, 16-bit word) to the following. (6 points)

binary _____ (straight base conversion)

octal **0** _____ (straight base conversion)

decimal _____ (convert to signed decimal)

Convert **-424** 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 **+361** 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)

$$\begin{array}{r} 11010111 \\ +10101001 \\ \hline \end{array}$$

N Z V C

| | | |

$$\begin{array}{r} 01000101 \\ +01111001 \\ \hline \end{array}$$

N Z V C

| | | |

$$\begin{array}{r} 01011011 \\ +10100101 \\ \hline \end{array}$$

N Z V C

| | | |

3. Branching (23 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 a and b have been
   properly declared as ints. */

if ( a > b )
{
    for ( a = 9951; b > a; ++a )
    {
        b = a % 71;
    }
}
else
{
    a = b - 35;
}
```

SPARC ASSEMBLY

```
! a is mapped to %13
! b is mapped to %16
```

4. Bit Operations / C Runtime Environment

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

```
set 0x9375ACD1, %l0
sra %l0, 15, %l0
```

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

```
set 0x9375ACD1, %l0
sll %l0, 9, %l0
```

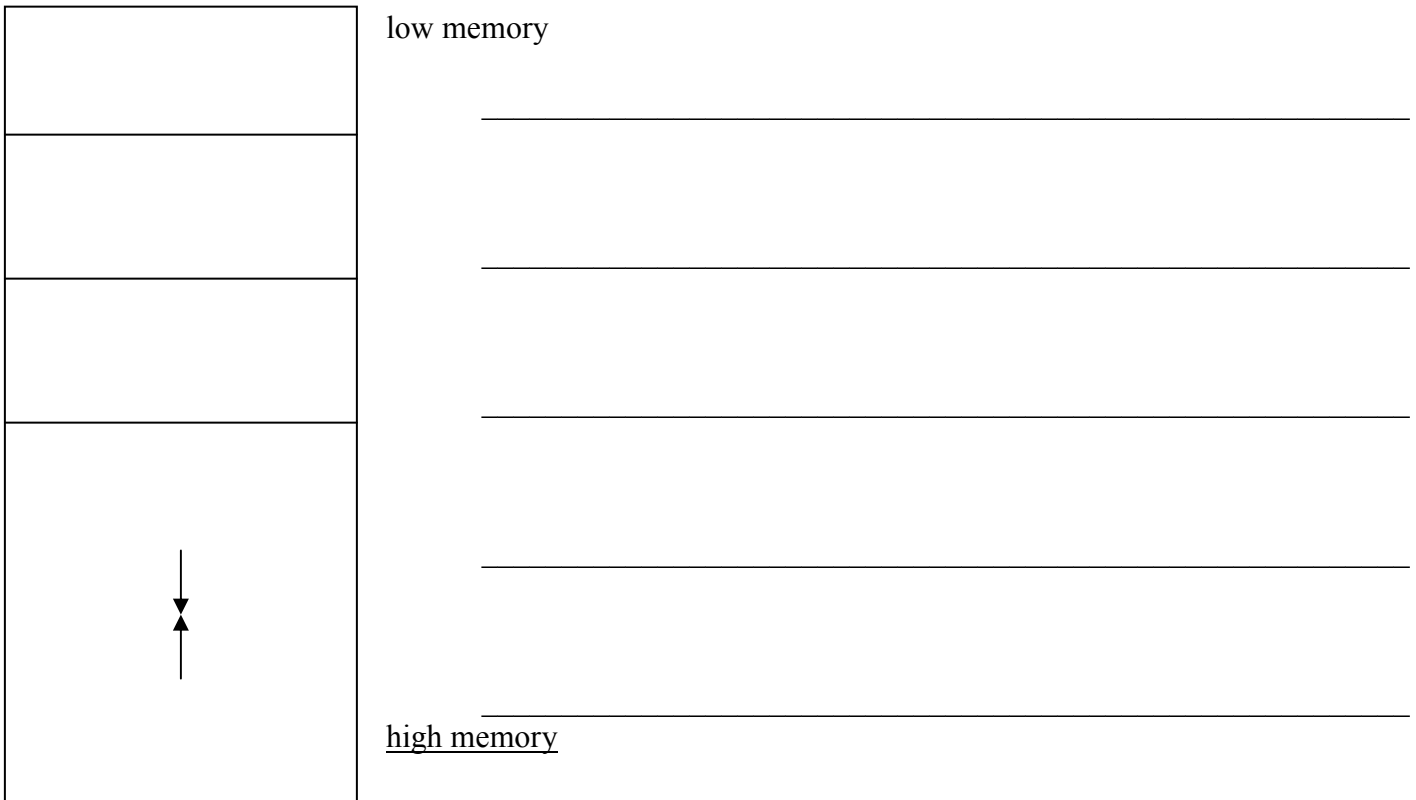
Value in %l0 is **0x**_____ (2 points)

```
set 0x9375ACD1, %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 Struct Variable)

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 */

short foo( char, int, unsigned short );

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

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

struct fubar {
    short      a;
    char       b[3];
    int        c;
    unsigned short d[2];
} fb; /* Local variable fb */

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

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

fb.a = foo( fb.b[2], fb.c, fb.d[0] );
```

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[3];
    int *local_stack_var2;

    local_stack_var2 = &local_stack_var1[1];    /* statement 1 */
    *local_stack_var2 = 8675309;                /* statement 2 */
    y = local_stack_var1[2];                    /* statement 3 */
    *local_stack_var2++ = x;                    /* statement 4 */
    return ( ++y + local_stack_var1[0] );       /* 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. (16 points)

SPARC assembly

```
.global      fubar
.section     ".text"
fubar:      /* Your unoptimized code goes below this point */
```

7. Load/Store/Memory Specify the hex values requested after those lines have been fully executed. (11 points)

```

.global main
.section ".data"
fmt: .asciz "0x%08X\n"      ! prints value as hex 0XXXXXXXXX
c:   .byte  0xAA
    .align 2
s:   .half  0xFADE
    .align 4
i1:  .word  0x87654321
i2:  .word  0x87654321
i3:  .word  0x87654321
x:   .word  0x66660000

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

set   x, %10
set   s, %11
lduh  [%11], %12           _____ Hex value in %12

stb   %12, [%10+3]        _____ Hex value in word labeled x

srl   %12, 4, %12         _____ Hex value in %12
stb   %12, [%10+1]

set   fmt, %0
ld    [%10], %01
call  printf              _____ Hex value in word labeled x
nop                                       (same as output of this printf)

set   i1, %10
set   c, %11
ldsb  [%11], %12         _____ Hex value in %12

sth   %12, [%10]
stb   %12, [%10+3]        _____ Hex value in word labeled i1

set   fmt, %0
ld    [%10], %01
call  printf              _____ Hex value in word labeled i1
nop                                       (same as output of this printf)

set   i2, %10
set   i3, %11
ld    [%11], %12         _____ Hex value in %12

stb   %12, [%10+1]        _____ Hex value in word labeled i2

sra   %12, 12, %12
sth   %12, [%10+2]        _____ Hex value in %12

set   fmt, %0
ld    [%10], %01
call  printf              _____ Hex value in word labeled i2
nop                                       (same as output of this printf)

ret
restore

```

Extra Credit (7 points)

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

```
#include <stdio.h>

int
main()
{
    char a[] = "DAT30";
    char *p = a;

    printf( "%c", *p++ );           _____
    printf( "%c", *(p+3) = *p);    _____
    printf( "%c", ++*p );          _____
    printf( "%c", **p );           _____
    printf( "%c", ++*p++ );        _____
    printf( "%d", ++p - a );       _____
    printf( "\n%s\n", a );         _____

    return 0;
}
```

A portion of the Operator Precedence Table

<u>Operator</u>	<u>Associativity</u>
++ postfix increment	L to R
-- postfix decrement	

* indirection	R to L
++ prefix increment	
-- prefix decrement	
& address-of	

* multiplication	L to R
/ division	
% modulus	

+ addition	L to R
- subtraction	

.	
.	
.	

= assignment	R to L

Scratch Paper

Scratch Paper