

Name _____

Signature _____

cs30x _____

Student ID _____

**CSE 30
Fall 2008
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	_____	(11 points)
SubTotal	_____	(102 points)
Extra Credit	_____	(5 points)
Total	_____	

1. Number Systems

Convert **0xFA6D** (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 **-462** 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 **+325** 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)

$\begin{array}{r} 11000101 \\ +00111001 \\ \hline \end{array}$	$\begin{array}{r} 01111111 \\ +00000001 \\ \hline \end{array}$	$\begin{array}{r} 11010111 \\ +10001001 \\ \hline \end{array}$
N Z V C	N Z V C	N Z V C
$\begin{array}{cccc} \hline & & & \\ \hline \end{array}$	$\begin{array}{cccc} \hline & & & \\ \hline \end{array}$	$\begin{array}{cccc} \hline & & & \\ \hline \end{array}$

3. Branching (20 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 use gotos/breaks/continues – just standard looping/conditional stmts.**

<u>SPARC ASSEMBLY</u>	<u>C</u>
<pre>.section ".text" baz: save %sp, -(92 + 8) & -8, %sp st %i0, [%fp - 4] mov %i0, %i0 set 5678, %i1 st %i1, [%fp - 8] cmp %i1, %i1 bge L1 nop L2: sll %i0, 4, %i1 st %i1, [%fp - 8] cmp %i0, %i1 be L3 nop add %i0, %i1, %i0 st %i0, [%fp - 4] ba L4 nop L3: sub %i0, %i1, %i0 st %i0, [%fp - 4] xor %i0, 0xDD, %i0 st %i0, [%fp - 4] L4: cmp %i1, %i1 bl L2 nop L1: ld [%fp - 8], %i0 ret restore</pre>	<pre>int baz(int a, int b) { int x; /* %fp - 4 and %i0 */ int y; /* %fp - 8 and %i1 */ }</pre>

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( char, unsigned short, int );

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

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

struct fubar {
    char          a;
    unsigned short b[2];
    char          c;
    int           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.a, fb.b[0], fb.d[1] );
```

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[4];
    int *local_stack_var2;

    *local_stack_var2 = 12345;          /* statement 1 */
    y = *local_stack_var2++;           /* statement 2 */
    local_stack_var1[0] = x;           /* statement 3 */
    local_stack_var2 = &local_stack_var1[1]; /* statement 4 */
    return ( y - local_stack_var1[3] ); /* 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 */
```

11. 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  0xDD
    .align 2
s:   .half  0xBABE
    .align 4
i1:  .word  0x12345678
i2:  .word  0x12345678
i3:  .word  0x12345678
x:   .word  0x77770000

.section ".text"
main:
save  %sp, -96, %sp
set   x, %10
set   s, %11
ldsh  [%11], %12           _____ Hex value in %12

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

srl   %12, 12, %12        _____ Hex value in %12
stb   %12, [%10+3]

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
ldub  [%11], %12          _____ Hex value in %12

sth   %12, [%10+2]        _____ Hex value in word labeled i1
stb   %12, [%10]

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

sth   %12, [%10]          _____ Hex value in word labeled i2

sra   %12, 16, %12        _____ Hex value in %12
stb   %12, [%10+1]

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 (5 points)

Optimize the following SPARC Assembly code fragment **by filling the nops in the delay slots**. 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 */

baz:
    save    %sp, -96, %sp

    mov     %i0, %l0
    set     5678, %l1

    cmp     %l1, %i1
    bge     L1
    nop

L2:
    sll     %i0, 2, %l1

    cmp     %l0, %l1
    be      L3
    nop

    add     %l0, %l1, %l0

    ba      L4
    nop

L3:
    sub     %i0, %i1, %l0
    xor     %l0, 0xFF, %l0

L4:
    cmp     %l1, %i1
    bl      L2
    nop

L1:
/* Other code you cannot use */
```

Optimized SPARC Assembly

Scratch Paper

Scratch Paper