CSE 30
Summer 2000
Final Exam

1. Number Systems ___________________  (17 points)
2. Binary Addition/Condition Code Bits/Overflow Detection ___________________  (12 points)
3. Branching ___________________  (20 points)
4. Bit Operations ___________________  (20 points)
5. Execution Speed & Compilation Sequence ___________________  (13 points)
6. Local Variables, The Stack, and Return Values ___________________  (22 points)
7. Subroutines ___________________  (22 points)
8. Floating Point ___________________  (12 points)
9. Linkage, Scope, Data ___________________  (32 points)
10. Machine Instructions ___________________  (20 points)
11. I/O & Virtual Memory ___________________  (22 points)
12. ALU & Control Unit ___________________  (14 points)
13. True or False ___________________  (16 points)
14. Programming Errors ___________________  (23 points)

SubTotal ___________________  (265 points)
Extra Credit ___________________  (12 points)
Total ___________________
1. Number Systems

Convert $\text{EDAD}_{16}$ (2’s complement, 16-bit word) to the following. (6 points)

octal _______________________________________

binary ______________________________________ 

decimal ______________________________________

Convert $-163_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (6 points)

sign-magnitude__________________________________________

1’s complement__________________________________________

2’s complement__________________________________________

Convert $+146_{10}$ to the following (assume 16-bit word). Express answers in hexadecimal. (3 points)

sign-magnitude__________________________________________

1’s complement__________________________________________

2’s complement__________________________________________

What are the values (as characters, not the ASCII values) of the following expressions:

char ch = ‘F’;
’a’ - ‘A’ + ch  ____________ (1 point)

char ch = ‘d’;
ch - ( ’a’ - ’A’ )  ____________ (1 point)
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)

\[
\begin{array}{ccc}
11010110 & +10111011 & \downarrow \\
+10111001 & +01010110 & +01101001 \\
\hline
\end{array}
\]

\[
N \quad Z \quad V \quad C
\quad N \quad Z \quad V \quad C
\quad N \quad Z \quad V \quad C
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
| & | & | & |
\hline
| & | & | & |
\hline
| & | & | & |
\hline
\end{array}
\]

3. Branching

Fill in the SPARC assembly instructions to perform the following statements. Do not optimize. (16 points)

\[
\text{int } x = 15; \quad \text{mov } 15, \%l0 \quad ! x \text{ mapped to } \%l0
\]

\[
\begin{array}{l}
\text{if ( } x \geq 15 \text{ ) } \{ \\
\quad \text{statement1;} \\
\quad \text{statement2;} \\
\quad ++x; \\
\} \text{ else } \{ \\
\quad \text{statement3;} \\
\quad \text{statement4;} \\
\quad --x;
\}
\end{array}
\]

\[
\text{/* statement___ */} \quad \text{/* statement___ */}
\]

\[
\text{/* statement___ */} \quad \text{/* statement___ */}
\]

For the following instruction sequence, mark with an X the conditional branch instructions which would transfer control to loop if used in place of ba. (4 points)

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>be</th>
<th>bne</th>
<th>bgu</th>
<th>bleu</th>
<th>ble</th>
<th>bge</th>
<th>bneg</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov -5, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subcc %10, -2, %10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4. Bit Operations

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

```assembly
set  0x87651234, %l0
set  0x87562134, %l1
and %10, %11, %10
```

Value in %l0 is _______________________________________ (2 points)

```assembly
set  0x87651234, %l0
sll %10, 7, %10
```

Value in %l0 is _______________________________________ (2 points)

```assembly
set  0x87651234, %l0
sra %10, 11, %10
```

Value in %l0 is _______________________________________ (2 points)

```assembly
set  0x56781234, %l0
set  0x51413121, %l1
btog %11, %10
```

Value in %l0 is _______________________________________ (2 points)

Write the unoptimized SPARC assembly function/module to extract the exponent bits from a single-precision floating point value passed in as a parameter and print the value of the real exponent. (12 points)

```assembly
.global printRealExponent /* void printRealExponent( float ); */
.section ".data"

.section ".text"
```
**5. Execution Speed & Compilation Sequence**

Considering the following different SPARC assembly language instructions, specify whether the code in column A will execute **slower/faster/same** relative to the code in column B. (6 points)

Pick one of:

- A slower than B
- A faster than B
- A same as B

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Column A</th>
<th>Column B</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld</td>
<td>[%l0], %l1</td>
<td>ld [%l1], %l0</td>
</tr>
<tr>
<td>add</td>
<td>%l2, %l1, %l3</td>
<td>add %l2, %l1, %l3</td>
</tr>
<tr>
<td>addcc</td>
<td>%l2, %l1, %l3</td>
<td>addcc %l2, -5, %l3</td>
</tr>
<tr>
<td>mov</td>
<td>4000, %o3</td>
<td>set 5000, %o3</td>
</tr>
</tbody>
</table>

Put the following in the correct order/sequence using the numbers to the left of each word: (7 points)

1. compiler
2. loader
3. assembler
4. execution
5. source code
6. linker
7. preprocessor

_____ → _____ → _____ → _____ → _____ → _____ → _____
6. Local Variables, The Stack, and Return Values

Here is a C function that allocates a local struct, performs some assignments and returns a value:

```c
char
fubar( int a[], int b ) {
    struct foo {
        int s1;
        char s2;
        short s3;
    } local_stack_var1;
    local_stack_var1.s1 = b;
    local_stack_var1.s2 = 'c';
    local_stack_var1.s3 = 12;
    a[3] = local_stack_var1.s1;
    return ( local_stack_var1.s2 );
}
```

Now write the entire equivalent **unoptimized** SPARC assembly language module/file to perform the equivalent. **You must allocate all local variables on the stack.** No short cuts. (22 points)

**SPARC assembly**
7. Subroutines

__________ subroutine is a subroutine that is expanded inline or is defined as a macro. (2 points)

__________ subroutine is a subroutine that does not call other subroutines. (2 points)

__________ subroutine is a traditional subroutine that passes parameters on the stack. (2 points)

__________ subroutine is a subroutine that supports recursion. (2 points)

__________ subroutine is a subroutine that increases generated code size. (2 points)

__________ subroutine is a subroutine that does not guarantee its args will only be evaluated once. (2 points)

__________ subroutine is a subroutine that uses save and restore instructions. (2 points)

__________ subroutine is the type of subroutine of which .mul is an example. (2 points)

__________ instruction slides the register set window down 16 registers. (2 points)

__________ instruction saves the current value of %pc in %o7. (2 points)

__________ instruction slides the register set window up 16 registers. (2 points)

8. Floating Point

Convert -26.375₁₀ (decimal fixed-point) to binary fixed-point (binary) and single-precision IEEE floating-point (hexadecimal) representations.

binary fixed-point __________________________________   (2 points)

IEEE floating-point __________________________________   (4 points)

Convert 0xC2D3C000 (single-precision IEEE floating-point representation) to fixed-point decimal.

binary fixed-point __________________________________   (2 points)

fixed-point decimal __________________________________   (4 points)
9. Linkage, Scope, Data

For the following program fragment, specify in which C runtime area/segment each of the following will be allocated: (24 points)

```c
int foo( void ); ______________ (where foo is define)

int x; ______________

int y = 5; ______________

static int a = 4; ______________

static int b; ______________

int main( int argc, char *argv[] ) { ______________ (argc & argv)

    static int e; ______________

    int *c; ______________

    int d = 9; ______________

    static int f = 8; ______________

    int (*g)(void); ______________ (where g is pointing)

    c = (int *) malloc( 12 ); ______________ (where c is pointing)

    g = foo; ______________ (where g is pointing)
}
```

Fill in the letter corresponding to the correct answer for each of the following:

A. loader B. linker C. assembler

Resolves address/symbols of external variables and subprograms ________ (2 points)

Generates machine code from assembly language ________ (2 points)

Copies executable binaries on disk to locations in memory ________ (2 points)

Takes .s files as input ________ (2 points)
10. Machine Instructions

Translate the following instructions into SPARC machine code. Use hexadecimal values for your answers. If an instruction is a branch, specify the number of instructions away for the target (vs. a Label).

```
add %l3, %i5, %o1 _______________________  (5 points)
ld  [%o1], %l4 _______________________  (5 points)
```

Translate the following SPARC machine code instructions into SPARC assembly instructions.

```
0x9224BFF6 _______________________  (5 points)
0x36800007 _______________________  (5 points)
```

11. I/O & Virtual Memory

Which combination of two of the following is fastest/most efficient for large data transfers.

Programmed I/O
Interrupt-driven _________________________ / _________________________ (4 points)
DMA
CPU blocked (Polling)

The _____________ translates ______________ to ______________ addresses. (6 points)
The _____________ caches these recently translated addresses. (2 points)

A ________________ cache ensures the data in the cache is the same as what is in main memory. (2 points)

Paging and swapping refer to parts of or entire program images being moved back and forth between what 2 storage areas/hierarchies? (4 points)

_________________________  and  _________________________

A __________ occurs because a page is needed but was not found to be in main memory. (2 points)

This type of I/O uses regular instructions to perform I/O and not any special instructions. (2 points)
12. ALU and Control Unit

Most RISC processors use some form of ____________________ control logic. (2 points)

The slower, yet more extensible types of Control Units are ___________________________. (2 points)

A device which can electronically connect and disconnect itself from the bus is called ______________________ (2 points)

Fill in the letter corresponding to the correct answer for each of the following:

_____ combinational logic/circuit (2 points)

_____ asynchronous (2 points)

_____ sequential logic/circuit (2 points)

_____ synchronous (2 points)

A. depends on timing by clock pulses or control signals
B. output is only a function of input
C. output is a function of input and some previous output (history)
D. depends only on input changes and not timing by clock pulses or control signals

13. True or False

T or F One way to generally speed up string compares on a Big-Endian machine is to use word compares instead of byte by byte compares. (2 points)

T or F With virtual memory, no more than one process using the same virtual address space can reside in physical memory at the same time. (2 points)

T or F Swapping will free up physical memory faster than paging. (2 points)

T or F The hidden bit in IEEE floating-point representation allows us to represent larger values. (2 points)

T or F When combining segmentation with paging, paging is done within segments. (2 points)

T or F Static linked binaries are usually larger than dynamic linked binaries. (2 points)

T or F Doubling the clock rate on a specific CPU will mean all instructions will execute twice as fast. (2 points)

T or F The resulting Text segment of a program compiled on a RISC system is generally smaller than that of the same program compiled on a CISC system. (2 points)
14. Programming Errors

There are several things wrong with this function such that it won’t compile and pass lint as it is. Put on your compiler and lint hat and list the problems/errors/things that are wrong which prevents this from passing lint and compiling. Assume the general logic is correct. (15 points)

```c
char * /* 932 -> "932" */
intToAscii( int num ) {
    char ascii[MAX_SIZE];
    int revNum = reverseNum( num );

    for ( i = 0; revNum != 0; ++i ) {
        /* Grab right-most digit, convert to char. */
        ascii[i] = (revNum % 10) + '0';
        revNum = revNum / 10;
    }
    ascii[i] = '\0';
    return( ascii );
}
```

Show the two ways we professed in the PAs to check for errors when calling strtol(). (8 points)

1. 

2. 


Extra Credit

What does the following SPARC assembly language program output?

```
.global main

.section "".data"

fmt:    .asciz  "%c"

.align 2

foo:    .half   0x5375, 0x7266, 0x204E, 0x616B, 0x646E, 0x0000

.section "".text"

main:
  save  %sp, -92 & -8, %sp
  clr   %l1
  set   foo, %l0
  ba    test
  nop

loop:
  set   fmt, %o0
  ldub  [%l0+%l1], %o1
  call  printf, 2
  nop
  inc   %l1

test:
  ldub  [%l0+%l1], %o1
  tst   %o1
  bne   loop
  nop
  set   fmt, %o0
  mov   0x0A, %o1
  call  printf, 2
  nop
  ret
  restore
```

Output ________________________
(4 points)

Now optimize the code to get the same result with the fewest cycles. You may not be able to eliminate all nop’s. Go for the fewest cycles assuming memory accesses are several more cycles than other instructions. (8 points)
<table>
<thead>
<tr>
<th>00 NUL</th>
<th>01 SOH</th>
<th>02 STX</th>
<th>03 ETX</th>
<th>04 EOT</th>
<th>05 ENQ</th>
<th>06 ACK</th>
<th>07 BEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>08 BS</td>
<td>09 HT</td>
<td>0A NL</td>
<td>0B VT</td>
<td>0C NP</td>
<td>0D CR</td>
<td>0E SO</td>
<td>0F SI</td>
</tr>
<tr>
<td>10 DLE</td>
<td>11 DC1</td>
<td>12 DC2</td>
<td>13 DC3</td>
<td>14 DC4</td>
<td>15 NAK</td>
<td>16 SYN</td>
<td>17 ETB</td>
</tr>
<tr>
<td>18 CAN</td>
<td>19 EM</td>
<td>1A SUB</td>
<td>1B ESC</td>
<td>1C FS</td>
<td>1D GS</td>
<td>1E RS</td>
<td>1F US</td>
</tr>
<tr>
<td>20 SP</td>
<td>21 !</td>
<td>22 &quot;</td>
<td>23 #</td>
<td>24 $</td>
<td>25 %</td>
<td>26 &amp;</td>
<td>27 '</td>
</tr>
<tr>
<td>28 (</td>
<td>29 )</td>
<td>2A *</td>
<td>2B +</td>
<td>2C ,</td>
<td>2D -</td>
<td>2E .</td>
<td>2F /</td>
</tr>
<tr>
<td>30 0</td>
<td>31 1</td>
<td>32 2</td>
<td>33 3</td>
<td>34 4</td>
<td>35 5</td>
<td>36 6</td>
<td>37 7</td>
</tr>
<tr>
<td>38 [</td>
<td>39 ]</td>
<td>3A :</td>
<td>3B ;</td>
<td>3C &lt;</td>
<td>3D =</td>
<td>3E &gt;</td>
<td>3F ?</td>
</tr>
<tr>
<td>40 {}</td>
<td>41 A</td>
<td>42 B</td>
<td>43 C</td>
<td>44 D</td>
<td>45 E</td>
<td>46 F</td>
<td>47 G</td>
</tr>
<tr>
<td>48 H</td>
<td>49 I</td>
<td>4A J</td>
<td>4B K</td>
<td>4C L</td>
<td>4D M</td>
<td>4E N</td>
<td>4F O</td>
</tr>
<tr>
<td>50 P</td>
<td>51 Q</td>
<td>52 R</td>
<td>53 S</td>
<td>54 T</td>
<td>55 U</td>
<td>56 V</td>
<td>57 W</td>
</tr>
<tr>
<td>58 X</td>
<td>59 Y</td>
<td>5A Z</td>
<td>5B [</td>
<td>5C \</td>
<td>5D ]</td>
<td>5E ^</td>
<td>5F _</td>
</tr>
<tr>
<td>60 `</td>
<td>61 a</td>
<td>62 b</td>
<td>63 c</td>
<td>64 d</td>
<td>65 e</td>
<td>66 f</td>
<td>67 g</td>
</tr>
<tr>
<td>68 h</td>
<td>69 i</td>
<td>6A j</td>
<td>6B k</td>
<td>6C l</td>
<td>6D m</td>
<td>6E n</td>
<td>6F o</td>
</tr>
<tr>
<td>70 p</td>
<td>71 q</td>
<td>72 r</td>
<td>73 s</td>
<td>74 t</td>
<td>75 u</td>
<td>76 v</td>
<td>77 w</td>
</tr>
<tr>
<td>78 x</td>
<td>79 y</td>
<td>7A z</td>
<td>7B {</td>
<td>7C</td>
<td>7D }</td>
<td>7E ~</td>
<td>7F DEL</td>
</tr>
</tbody>
</table>
Scratch Paper