CISE Qualifying Exam I
5 January 2015

STUDENT EXAM NUMBER: __________

General Instructions:

1. This is a closed book exam. You have 6 hours to complete the examination. The proctor will tell you when to begin and when to end.

2. To pass the exam, you must successfully complete at least four (4) of the six (6) areas. You may attempt more than 4 areas, but you do not need to do so.

3. Print your student exam number at the top right corner of every answer page submitted. No names or SU ID numbers should appear on any of the pages you submit.

4. Start each area on a separate page. If an area involves multiple parts, you may answer multiple parts on the same page. Unless specified otherwise, you should answer all parts of a given area.

5. When you have finished your exam, please indicate below the areas that you are submitting answers, along with the number of pages you’re submitting for each answer.

<table>
<thead>
<tr>
<th>Submitting Answers</th>
<th># of pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
</tr>
<tr>
<td>Architecture/Computer Organization</td>
<td></td>
</tr>
<tr>
<td>Data Structures &amp; Programming</td>
<td></td>
</tr>
<tr>
<td>Digital Logic Circuit Design</td>
<td></td>
</tr>
<tr>
<td>Operating Systems</td>
<td></td>
</tr>
<tr>
<td>Programming Languages</td>
<td></td>
</tr>
</tbody>
</table>
Algorithms

BACKGROUND: Suppose that:

(i) $G = (V = \{1, \ldots, n\}, E)$ is a directed acyclic graph represented by adjacency lists.

(ii) $G$'s vertices are in a topological-sort order, i.e., $(u, v) \in E \implies u < v$.

(iii) Each $(u, v) \in E$ has a positive length $\ell(u, v)$.

Note that from our assumptions above, 1 is a source node, i.e., it has in-degree 0. Define the function $\text{sumLens}_G : \{2, \ldots, n\} \to \mathbb{N}$ as follows:

$$\text{sumLens}_G(v) = \sum \{ \ell(s, t) : (s, t) \text{ is an edge on some } G\text{-path from } 1 \text{ to } v \}.$$

EXAMPLE: For $G$, the dag on below, $\text{sumLens}_G(6) = 335$.

Part I:

(a) (15%) QUESTION: Give a $O(|V| + |E|)$ time algorithm for computing $\text{sumLens}_G$.

(b) (10%) Justify the $O(|V| + |E|)$ run-time.

(c) (15%) Clearly explain why your algorithm is correct.

Part II:

(d) (20%) Give a $O(|V| + |E|)$-time algorithm that determines, for each $v \in \{2, \ldots, n\}$, the length of a shortest path from 1 to $v$. (If there is no path from 1 to $v$, the distance is $\infty$.)

EXAMPLE: For $G$, the dag above, $\text{shortest}[2] = \infty$, $\text{shortest}[3] = 2$, $\text{shortest}[4] = 3$, $\text{shortest}[5] = 12$, and $\text{shortest}[6] = 112$.

(e) (10%) Justify the $O(|V| + |E|)$ run-time.

(f) (20%) Clearly explain why your algorithm is correct.

(g) (10%) If we allowed negative length edges, does your algorithm still work? Why or why not?
Question 1  Consider the following assembly program executed on a typical MIPS CPU with 5-stage (i.e., IF stage, ID stage, EX stage, MEM stage, and WB stage) pipeline:

```
Loop:  LW  r2, 0(r2)
       BEQ r2, r0, LOOP;
       OR  r2, r2, r3
       SW  r2, 0(r5)
```

Assume that there is no delay slot and the branch instruction is executed in the ID stage. Also assume that the loop is repeated for 100 iterations and the instructions before and after this program are NOPs.

(a) Draw the pipeline diagram for the first 2 iterations.

(b) How many clock cycles does it take to execute the entire assembly program, given that the loop is iterated for 100 times. This should be the duration starting from the time that the first instruction of the program enters the pipeline, to the time that the last instruction (i.e. SW r2, 0(r5) ) leaves the pipeline.

(c) If a one bit branch predictor is used, which is initialized to “N” (i.e. branch not taken), how many correct and wrong predictions that it will produce? How many clock cycles will it take to execute the above code using this predictor?

Question 2  Assume 4 KB pages, a 4-entry 2-way set associative TLB, and LRU replacement. When multiple physical pages are available, the OS will first use the page with the lowest index. The following shows the current content in page table.

<table>
<thead>
<tr>
<th>Virtual Page #</th>
<th>Physical Page/Disk</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Disk</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Disk</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Disk</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Disk</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Disk</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>1</td>
</tr>
</tbody>
</table>
The TLB currently has two valid entries, which stores the information of virtual pages 11 and 3. Page 11 was loaded into main memory earlier than page 3. The other two entries of TLB are not currently used.

**QUESTION:** Suppose we have the following virtual address stream

0x123D, 0x8B3, 0x365C, 0x871B, 0xBE82, 0x3140

For each memory reference, indicate if it is a hit in the TLB, a hit in the page table, or a page fault. Also show the final state of the TLB and page table.

**Question 3** Consider the following C code:

```c
int A[8][8];
int B[8][8];
int C[8][8];
int i, j, k;

for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
        C[i][j] = 0;
        for (k = 0; k < 8; k++) C[i][j] = C[i][j] + A[i][k] * B[k][i];
    }
}
```

Assume 2-way set associative cache and its capacity is 32 KB. The size of cache block is 64 Byte. The starting addresses of arrays A, B and C are 0x0010, 0x0110, and 0x0210.

(a) Estimate the data cache miss rate of the above code.

(b) Assume that write back policy with write allocation is used for the data cache. Also assume that the data cache hit time is 1ns, the main memory read bandwidth is 1GB/sec and write bandwidth is 0.5GB/sec. The main memory access latency for both read and write is 100ns. Estimate the average data memory access time.
Data Structures & Programming

BACKGROUND: This problem concerns binary trees, such as the ones pictured below. Each tree-node has the fields:

- **label**: a 32-bit integer
- **left**: a pointer to the root of the node’s left subtree
- **right**: a pointer to the root of the node’s right subtree

A **binary search tree** (abbreviated: *BST*) is a binary tree with the property that, for each tree node *nd* with label *k*:

- the elements in *nd*’s left subtree have labels < *k* and
- the elements in *nd*’s right subtree have labels ≥ *k*.

Our BST’s will have no repeated values.

EXAMPLES: *t1* is a BST, but *t2* is not because 21 is in the left subtree of the 20-node.

Provide C, C++, Haskell, Java, or ML code to answer the following questions. Clarity counts—if your answers are too hard to follow, they will be marked wrong.

All the questions have short, straightforward answers.

QUESTIONS:

(a) (10%) Give a data-structure for representing tree-nodes as described above.

(b) (15%) Write a function that, given a binary tree, returns the post-order sequence of integer values in the tree. The sequence can be represented by a list or an array. The function should run in *O*(*n*)-time where *n* is number of nodes in the tree. **EXAMPLE**: For *t1* the function would return the sequence [5, 17, 11, 34, 42, 38, 20].

(c) (15%) Write a function that inserts a value, *v*, into a BST, *t*, so that updated tree is a BST. When *t* already contains *v*, then *t* is unchanged. The function should run in *O*(h)-time where *h* is the height of *t*.

(d) (30%) Write a function that, given a binary tree *t*, tests if *t* is a BST, i.e., it returns *true* if it is, and *false* if it is not. The function should run in *O*(n)-time where *n* is number of nodes in *t*.

(e) (30%) Write a function that, given a sequence of distinct integer values (in a list or an array) that results from a postorder traversal of a BST *t*, rebuilds *t* from the sequence. The function should run in *O*(h·n) time, where *h* is the height of *t* and *n* is the length of the sequence. **EXAMPLE**: Given the input sequence [5, 17, 11, 34, 42, 38, 20], the function would produce tree *t1* above. Do not worry about the case where the input does not come from a postorder traversal of a BST.
Problem 1: Controller Design (Finite-State Machine Design)

Implement the following use-case description using combinational logic and a state register.

**Use case description:** Create a completely specified deterministic three-state FSM with a single input \( x \) and a single output \( y \) with the following behavior:

1. The initial state is \( S_0 \). If the input \( x \) is 0, the machine changes to state \( S_1 \). If the input \( x \) is 1, the next state of the machine changes to state \( S_2 \). The output \( y \) is 0 in state \( S_0 \).

2. If the machine is in state \( S_1 \), the output \( y \) is 0. If the input \( x \) is 0, the next state is \( S_0 \). If \( x \) is 1, the next state is \( S_2 \).

3. If the machine is in state \( S_2 \), the output \( y \) is 1. If the input \( x \) is 0, the next state is \( S_2 \). If \( x \) is 1, the next state is \( S_0 \).

Fully document each stage of your design by including:

1. A state-transition diagram,

2. A truth table whose inputs are the current machine state and input variables, and whose outputs are the next-state functions and output functions, and

3. Logical formulas for next-state and output functions specifying the combinational logic necessary to implement the machine.

Use the standard binary coding for your state encoding, i.e.,

\[
S_0 = s_1s_0 = 00, \quad S_1 = s_1s_0 = 01, \quad S_2 = s_1s_0 = 10
\]

**Important:** For the unused state encoding \( s_1s_0 = 11 \), regardless of input value the next state should be \( S_0 \) and the output \( y \) should be 0.
(a) State-Transition Diagram

(b) Next-state and output truth table

<table>
<thead>
<tr>
<th>Present State, Input</th>
<th>Next State, Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 )</td>
<td>( s_0 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
(c) Implementation using a state register and combinational logic

For the architecture shown below, based on the next-state and output truth table, write the logical formulas that define \( n_1(s_1, s_0, x) \), \( n_0(s_1, s_0, x) \), and \( y(s_1, s_0) \). Note: minimization is not necessary.

\[
\begin{align*}
n_1(s_1, s_0, x) &= \\
n_0(s_1, s_0, x) &= \\
y(s_1, s_0) &=
\end{align*}
\]
(c) Implementation using a state register and combinational logic

For the architecture shown below, based on the next-state and output truth table, write the logical formulas that define $n_1(s_1, s_0, x), n_0(s_1, s_0, x)$, and $y(s_1, s_0)$. Note: minimization is not necessary.

$$n_1(s_1, s_0, x) =$$

$$n_0(s_1, s_0, x) =$$

$$y(s_1, s_0) =$$
Problem 2: Data Paths and Control Paths

Fill in the table below to implement the following microprogrammed operations to place the value $A + B$ into $Reg_3$. (Note: $X$ is don’t care or don’t know. 1 enables or loads values onto a bus or into a register)

1. $Reg_0 \leftarrow A$
2. $Reg_0 \leftarrow B; \quad Reg_1 \leftarrow Reg_0$
3. $Reg_2 \leftarrow Reg_0$
4. $Reg_3 \leftarrow Reg_1 + Reg_2$

<table>
<thead>
<tr>
<th>Clock Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$In$</td>
<td>A</td>
<td>B</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$Reg_0$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Reg_1$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Reg_2$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Reg_3$</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$LD_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$LD_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$LD_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$LD_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$EN_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$EN_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$EN_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$EN_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$F_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>$F_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Operating Systems

Part I. Memory Management (30%) For a byte-addressable computer architecture with segmentation with multi-level paging, a page size of 4KB, and 64-bit virtual and physical addresses, answer the following questions.

1. (10%) How many bits are required to address the offset within a page and why?

2. (20%) Assume there are 16 segments and the most significant bits in the virtual address format are used to refer to segments 0 to 15. Also, assume that one page table entry requires 16 bytes and we require that each page table fits into a single page. How many levels of page tables would be required to completely map the 64-bit virtual address space? Show the virtual address format by showing how all of the 64 bits are used.

Part II. Unix Processes (40%) The following table is from Tanenbaum explaining some Unix system calls related to process management.

<table>
<thead>
<tr>
<th>System Calls</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid = fork()</td>
<td>Create a child process identical to the parent</td>
</tr>
<tr>
<td>pid = waitpid(pid)</td>
<td>Wait for a child to terminate</td>
</tr>
<tr>
<td>s = execl(name, argv, environs)</td>
<td>Replace a process' core image</td>
</tr>
<tr>
<td>exit(status)</td>
<td>Terminate process execution and return status</td>
</tr>
</tbody>
</table>

1. (20%) Consider the following program.

```c
#include <stdio.h>
#include <unistd.h>

main (int argc, char** argv) {
    int proc_id = fork();
    int x = 5;
    if (proc_id == 0) {
        x = x + 5;
    } else {
        proc_id = fork();
        x = x + 10;
        if (proc_id) {
            x = x + 7;
        }
    }
    printf("Value of x = %d\n", x);
    return (0);
}
```

(a) How many processes are created by this program and why?
(b) Each time we run the program, these processes can run in a different order depending on the scheduler's behavior. Show all possible outputs of the program and explain your answers.

2. (20%) Consider the following program. There are two processes: the parent and the child. Note: The 'execl("/bin/date","date",0);' system call will execute the 'date' program that displays the current time and date.

```c
1. #include <stdio.h>
2. #include <unistd.h>
3. main (int argc, char ** argv) {
4. int val = 5;
5. int proc_id;
6. printf("Hello!\n");
7. proc_id = fork();
8. if (proc_id)
9. waitpid(proc_id);
10. else {
11. printf("Value of val = %d\n", val);
12. execl("/bin/date","date",0);
13. }
14. val++;
15. printf("Goodbye! Value of val = %d\n", val);
16. return(val);
17. }
```

(a) Which line of the code is executed as the first line by the child process and why?

(b) Show the output of the program and explain your answer.

Part III. File System Design (30%) LFS (the Log-structured file system) is designed to take advantage of increased CPU speed and memory size; it also considers the fact that improvements on mechanical hard disk have been mostly on cost per unit storage. As the capacities of solid state drives (SSDs) increase, some computers replace mechanical hard disk drives with SSDs. SSDs do not have mechanical moving parts. Consider the following specification of Intel 520 Cherryville SSD. The SSD also has the following additional characteristics: (1) For random writes, it needs to erase a large number of pages (32 to 64) and reprogrammed. (2) Memory cells have burnout characteristic; that is, each cell has limited erase/program cycles ranging from 1,000 to 100,000 writes.

<table>
<thead>
<tr>
<th>Hard Disks</th>
<th>Capacity</th>
<th>Sequential R/W</th>
<th>Latency (Read)</th>
<th>Latency (Write)</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel SSD</td>
<td>240GB</td>
<td>550 MB/s</td>
<td>0.04ms</td>
<td>0.2ms</td>
<td>$0.06/GB</td>
</tr>
</tbody>
</table>

1. Explain how LFS may or may not be suitable for a computer system with the SSD. You must justify your answer.
Programming Languages

The syntax for a simple language of arithmetic expressions appears at the bottom of this page. Two separate operational semantics for this language appear on the following page.

Consider the following claim:

For all arithmetic expressions \( E \), states \( \sigma, \sigma' \), and integers \( n \),

\[
\text{if } \langle E, \sigma \rangle \Rightarrow \langle n, \sigma' \rangle, \text{ then there is a state } \sigma'' \text{ such that } \langle E, \sigma \rangle \Rightarrow^* \langle n, \sigma'' \rangle.
\]

Is this claim true? Provide rigorous support for your answer as follows:

- If you answered **YES**: Give a proof by induction (e.g., structural induction, rule induction, or induction on the derivations).

- If you answered **NO**: Give specific instances of \( E, \sigma, \sigma', n \) for which the property fails to hold. Give a complete derivation of \( \langle E, \sigma \rangle \Rightarrow \langle n, \sigma' \rangle \) and provide convincing support for the lack of a corresponding derivation for \( \langle E, \sigma \rangle \Rightarrow^* \langle n, \sigma'' \rangle \).

Syntax  

The syntax relies on the following sets:

- **Num**, the set of integer numerals, ranged over by meta-variable \( n \)
- **Var**, the set of variables, ranged over by meta-variable \( x \)
- **AExp**, the set of arithmetic expressions, ranged over by meta-variable \( E \)

Based on these sets (and the stated conventions for meta-variables), the abstract syntax for the set **AExp** is defined by the following grammar:

\[
E \ ::= \ n \ | \ x \ | \ \text{inc} \ x \ | \ E_1 + E_2 \ | \ \text{pick}(E_1, E_2)
\]
Semantics  For the semantics, we make use of the following fairly standard conventions:

- We use $n$ as a meta-variable ranging over the set $\mathbb{Z}$ of integer values.
- Numerals are distinguished from integer values by underlining. For example, $\underline{3}$ is a numeral (i.e., syntax), whereas $3$ denotes a semantic value.
- Likewise, $\text{plus}$ denotes addition over integer values, in contrast to $+$ (which is syntax).
- A state $\sigma$ is a finite partial function from variables to integer values:
  
  $$\sigma : \text{Var} \to \mathbb{Z}.$$  

  Thus, for any variable $x$, $\sigma(x)$ is the value associated with $x$.
- $\sigma[v/x]$ denotes the state that is identical to $\sigma$ except for mapping $x$ to $v$.

Original Semantics ($\Rightarrow$):

$$\text{Num} \quad \langle n, \sigma \rangle \Rightarrow \langle n, \sigma \rangle \quad \text{Var} \quad \langle x, \sigma \rangle \Rightarrow \langle n, \sigma \rangle \quad \sigma(x) = n$$

$$\text{Inc} \quad \langle x, \sigma \rangle \Rightarrow \langle n, \sigma[(n+1)/x] \rangle \quad \sigma(x) = n$$

$$\text{Add} \quad \langle E_1, \sigma \rangle \Rightarrow \langle n_1, \sigma_1 \rangle \quad \langle E_2, \sigma_1 \rangle \Rightarrow \langle n_2, \sigma_2 \rangle \quad n = \text{plus}(n_1, n_2)$$

$$\text{Choice} \quad \langle E_1, \sigma \rangle \Rightarrow \langle n_1, \sigma_1 \rangle \quad \langle E_2, \sigma \rangle \Rightarrow \langle n_2, \sigma_2 \rangle \quad k \in \{1, 2\}$$

Alternate Semantics ($\Rightarrow^\text{ALT}$):

$$\text{aNum} \quad \langle n, \sigma \rangle \Rightarrow^\text{ALT} \langle n, \sigma \rangle \quad \text{aVar} \quad \langle x, \sigma \rangle \Rightarrow^\text{ALT} \langle n, \sigma \rangle \quad \sigma(x) = n$$

$$\text{aInc} \quad \langle x, \sigma \rangle \Rightarrow^\text{ALT} \langle n, \sigma[(n+1)/x] \rangle \quad \sigma(x) = n$$

$$\text{aAdd} \quad \langle E_1, \sigma \rangle \Rightarrow^\text{ALT} \langle n_1, \sigma_1 \rangle \quad \langle E_2, \sigma_1 \rangle \Rightarrow^\text{ALT} \langle n_2, \sigma_2 \rangle \quad n = \text{plus}(n_1, n_2)$$

$$\text{aChoice} \quad \langle E_1, \sigma \rangle \Rightarrow^\text{ALT} \langle n_1, \sigma_1 \rangle \quad \langle E_2, \sigma \rangle \Rightarrow^\text{ALT} \langle n_2, \sigma_2 \rangle \quad k \in \{1, 2\}$$

$$\langle \text{pick}(E_1, E_2), \sigma \rangle \Rightarrow^\text{ALT} \langle n, \sigma \rangle$$