Spring 2020 Q-exam, CSCE 513, Architecture

Question 1:

You have a processor that runs with a 2.5 GHz clock.

You have an L1 instruction cache and an L1 data cache, but no L2 cache (to keep this problem simpler).
That is, the L1 cache connects directly to memory.

Assume the read request to the L1 instruction cache is prefetched/fully pipelined so the cost is zero for getting “the next'” instruction to the processing unit, if that instruction is in the L1 cache.
An L1 instruction cache miss happens 15% of the time; that is, 15% of the time the instruction isn't in L1 instruction cache.

Assume the cost of getting one word from L1 data cache is one clock cycle.

Assume that 20% of the instructions are LOAD and 10% are STORE.

All cache blocks are 16 bytes.
All cache block transfers are one block of 16 bytes.

Assume that the cost of getting one cache block from main memory to the L1 cache (data or instruction) is 80 nanoseconds plus one block transfer of 16 bytes over a 133MHz bus.

Assume that the L1 data cache is write-through, that there are no buffers, and that L1 data cache blocks are dirty 40% of the time.

1. What's the average cost in nanoseconds of fetching instructions?

2. What's the average cost in nanoseconds of a data read?

3. What's the average cost in nanoseconds of a data write?

Question 2:

Describe the basic argument for why each of the following ideas is/was considered an idea that would improve computer performance, and explain briefly the characteristic of computer workload that would be necessary in order for this to be an idea that actually worked.

1. The Tera Computer Corporation hardware withparalleldatastreams and multiple independent logic units.
2. A vector computer like the CDC 6400, CDC 6600, CRAY-1, CRAY- 2, CRAY-XMP, CRAY-YMP.
3. A memory hierarchy that includes cache memory.
4. Pipelining of instruction processing.

Question 3:

Explain the problem of cache coherence in a multiprocessor. Describe at least one method for solving the problem of maintaining cache coherence.

Spring 2020 CSE Qualifying Exam
CSCE 531, Compilers

1. **LR-Parsing.** Consider the following augmented grammar G with start symbol S':

$$\begin{array}{rcl} S' & \rightarrow & S \\ S & \rightarrow & V \text{ `=' } E \\ S & \rightarrow & E \\ S & \rightarrow & V \\ V & \rightarrow & \textbf{id} \\ V & \rightarrow & \text{`*' } E \\ E & \rightarrow & V \end{array}$$

   (a) For the grammar G above generate all of the LR(1) sets of items $I_0, I_1, I_2, \ldots, I_9$ along with complete transition information for an LALR parser.

   (b) Using the sets-of-items constructed in part (b), construct the action table and describe any conflicts. Assume the productions are numbered in order from 0 to 6.

   (c) Describe in detail how an arbitrary LR parsing algorithm will proceed in general when the next token is t and the stack contents are $s_0, s_1, \ldots, s_{top-1}, s_{top}$.

2. **A Syntax-Directed Definition.** The "break" statement of C causes the exiting of an enclosing loop or switch statement. It should generate an unconditional jump to some appropriate label. Given a language with the statement types below:

$$\begin{array}{rcl} \langle \textit{While} \rangle & ::= & \textbf{while (Boolean) do} \langle \textit{StatementList} \rangle \textbf{endwhile;} \\ \langle \textit{IfThenElse} \rangle & ::= & \textbf{if (Boolean) then} \langle \textit{StatementList} \rangle_1 \textbf{ else} \langle \textit{StatementList} \rangle_2 \textbf{ endif;} \\ \langle \textit{Break} \rangle & ::= & \textbf{break;} \\ \langle \textit{Assignment} \rangle & ::= & \textbf{var= expression;} \end{array}$$

   (a) (20% credit) Provide productions for $\langle \textit{StatementList} \rangle$ and $\langle \textit{Statement} \rangle$. The former produces one or more statements, and the latter produces a while, if-then-else, break, or assignment statement.

(b) (80% credit) Add enough semantic actions to the grammar to handle control-flow caused by the break statement *only*.

You may assume any attributes, data structures, or supporting routines that you find useful, provided you make it reasonably clear how they behave. Both synthesized and inherited attributes are allowed.

3. Consider the intermediate code below.

```
 1  start:  x := 1
 2          y := 1
 3          sum := y
 4          sum := sum + 1
 5  L1:     if x = n then goto E2
 6  L2:     if y = x then goto E1
 7          t1 := a[y]
 8          t2 := y * t1
 9          t3 := t1 + x
10          if t3 < n then goto S
11          t3 := t3 - n
12  S:      sum := sum + t3
13          y := y + 1
14          goto L1
15          goto L2
16  E2:     a[x] := sum
17          sum := x + 1
18          x := x + 1
19          goto L2
20  E1:     x := x - 1
21          t1 := a[x]
22          print t1
23          if x = 0 then goto out
24          goto E2
25  out:    no-op
```

Assume that there are no entry points into the code from outside other than at the start.

(a) (20% credit) Decompose the code into basic blocks B1,B2, ..., giving a range of line numbers for each.

(b) (20% credit) Draw the control flow graph, and describe any unreachable code.

(c) (40% credit) Fill in an 25-row table listing which variables are live at which control points. Treat `a` as a single variable. Assume that `n` and `sum` are the only

live variables immediately before line 25 (the only exit point). Your table should look like this:

Before line	**Live variables**
1	$\cdots$
2	$\cdots$
$\cdots$	$\cdots$
25	$\cdots$

(d) (20% credit) Describe any simplifying transformations that can be performed on the code (i.e., transformations that preserve the semantics but reduce (i) the complexity of an instruction, (ii) the number of instructions, (iii) the number of branches, or (iv) the number of variables).

Spring 2020 Q-exam — CSCE 750 (Algorithms)

1. **(Solving a Recurrence)** Let $T(n)$ be the function defined for all integers $n \geq 1$ by the following recurrence:

$$T(n) = n + \sum_{i=1}^{\lceil \sqrt{n} \rceil - 1} T(i)$$

Find an expression $f(n)$, as simple as possible, such that $T(n) = \Theta(f(n))$. Use the substitution method to **prove** that your answer is correct.

2. **(Threading a BST)** Suppose that in a certain implementation of binary search trees, each node x has four attributes: $x.key$ (a number), $x.left$, $x.right$, and $x.next$. As usual, $x.left$ and $x.right$ point to the left and right subtrees of x, respectively. Also as usual, all the keys k in $x.left$ satisfy $k < x.key$, and all the keys ℓ in $x.right$ satisfy $\ell > x.key$ (it follows that there are no duplicate keys). Also as usual, a BST T is accessed via a pointer to its root if T is nonempty, or by a NULL pointer if T is empty.

   (a) (25%) **Describe** an algorithm that takes a BST T as input, sets the (uninitialized) *next* attribute of each node x in T to point to the node in T with the least key that is larger than $x.key$, or NULL if x has the largest key in T. Your algorithm should also return a pointer to the node in T with the smallest key, or NULL if T is empty.

   Note that nodes do not have *parent* attributes. Your algorithm effectively augments a BST with a sorted linked list through all its nodes, returning a pointer to the first node in the list.

   Explain why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct.

   Your algorithm should run in $O(n)$ time, where n is the number of nodes in the tree.

   (b) (75%) Assuming that the *next* attributes of T have been correctly set as in part (a), describe how to modify the usual BST INSERT operation to maintain the linked list after inserting a new node into the tree. If the key you are inserting is already in the tree, your operation should do nothing to the tree. Your operation should return a pointer to the newly inserted node or NULL if no insertion took place.

   Your modified INSERT operation should run in time $O(d)$, where d is the depth of the tree.

   You should not perform any rebalancing of the tree.

3. Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of natural numbers (including 0). For any directed graph $G = (V, E)$, a *Grundy numbering* of G is a map $g : V \to \mathbb{N}$ such that for every vertex $u \in V$, $g(u)$ is the least natural number *not* in the set $\{g(v) : (u, v) \in E\}$. So for example, if u has outdegree 0, then $g(u) = 0$.

   (a) Every directed *acyclic* graph has a unique Grundy numbering. Describe an algorithm that computes it. For every $u \in V$, your algorithm should set the attribute $u.g$ to $g(u)$. You may assume that G is represented as usual by an array $V[1 \ldots n]$ of adjacency lists, where $V = \{v_1, \ldots, v_n\}$. You may also assume that G is topologically sorted, i.e., for all $1 \leq i, j \leq n$, if $(v_i, v_j) \in E$, then $i < j$.

   Your algorithm should run in time $O(n + m)$, where $n = |V|$ and $m = |E|$.

   (b) Give two reasonably simple examples G_1 and G_2 of directed graphs, where G_1 has no Grundy numbering and G_2 has at least two different Grundy numberings.