CSE Qualifying Exam, Spring 2008

February 2, 2008
Architecture

1. You are building a system around a processor with in-order execution that runs at 1.1 GHz and has a CPI of 0.7 excluding memory accesses. The only instructions that read or write data from memory are loads (20% of all instructions) and stores (5% of all instructions).

The memory system for this computer is composed of a split L1 cache that imposes no penalty on hits. Both the I-cache and D-cache are direct mapped and hold 32 KB each. The I-cache has a 2% miss rate and 32-byte blocks and the D-cache is write through with a 5% miss rate and 16-byte blocks. There is a write buffer on the D-cache that eliminates stalls for 95% of all writes.

The 512 KB write-back, unified L2 cache has 64-byte blocks and an access time of 15 ns. It is connected to the L1 cache by a 128-bit data bus that runs at 266 MHz and can transfer one 128-bit word per bus cycle. Of all memory references sent to the L2 cache in this system, 80% are satisfied without going to main memory. Also, 50% of all blocks replaced are dirty.

The 128-bit-wide main memory has an access latency of 60 ns, after which any number of bus words may be transferred at the rate of one per cycle on the 128-bit-wide 133 MHz main memory bus.

(a) What is the average memory access time for instruction accesses?
(b) What is the average memory access time for data reads?
(c) What is the average memory access time for data writes?
(d) What is the overall CPI, including memory accesses?

2. A reduced hardware implementation of the classic five-stage RISC pipeline might use the EX stage hardware to perform a branch instruction comparison and then not actually deliver the branch target PC to the IF stage until the clock cycle in which the branch instruction reaches the MEM stage. Control hazard stalls can be reduced by resolving branch instructions in ID, but improving performance in one respect may reduce performance in other circumstances. How does determining branch outcome in the ID stage have the potential to negatively affect performance?

3. Three enhancements with the following speedups are proposed for a new architecture:

   Speedup1 = 30
   Speedup2 = 20
   Speedup3 = 15

Only one enhancement is usable at a time.

(a) If enhancements 1 and 2 are each usable for 25% of the time, what fraction of the time must enhancement 3 be used to achieve an overall speedup of 10?
(b) Assume the enhancements can be used 25%, 35%, and 10% of the time for enhancements 1, 2, and 3, respectively. For what fraction of the reduced execution time is no enhancement in use?
Theory

1. Fix a finite alphabet \( \Sigma \). For any language \( L \subseteq \Sigma^* \) define

\[
\text{rotate}(L) = \{wa \mid a \in \Sigma \text{ and } w \in \Sigma^* \text{ and } aw \in L\}.
\]

Show that if \( L \) is regular, then \( \text{rotate}(L) \) is regular.

[Hint: First show that if \( L \) is regular, then for any fixed \( a \in \Sigma \), the language

\[
L_a := \{w \mid w \in \Sigma^* \text{ and } aw \in L\}
\]

is regular.]

2. We use the standard model of a Turing machine with a one-way infinite tape whose cells are labeled 0, 1, 2, \ldots starting with the leftmost cell. For \( t \geq 0 \), we say that TMs \( M_1 \) and \( M_2 \) are separate at time \( t \) iff after exactly \( t \) steps since starting the two machines are scanning cells with different labels, i.e., after exactly \( t \) steps \( M_1 \) is scanning some cell \( i \), \( M_2 \) is scanning some cell \( j \), and \( i \neq j \). Say that \( M_1 \) and \( M_2 \) eventually separate iff there is a time \( t \) at which \( M_1 \) and \( M_2 \) are separate.

Let

\[
L = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs that eventually separate on input } \varepsilon\}.
\]

(a) (40%) Explain why \( L \) is Turing-recognizable.

(b) (60%) Show that \( L \) is undecidable.

3. Show that the language

\[
L = \{\langle M, 1^n \rangle \mid M \text{ is a DFA, } n \geq 0, \text{ and } M \text{ accepts at least one string of length } n\}
\]

is in P.
Algorithms

1. Using the tree method, give tight asymptotic bounds for the function $T(n)$ that satisfies the recurrence $T(n) = 2T(n/2) + n \log n$. Show your work.

2. In a certain kind of augmented binary search tree, each node $v$ has four attributes:

   - $key[v]$: the actual value at the node $v$ (for simplicity, assume that this is just a number),
   - $left[v]$: the pointer to $v$’s left subtree,
   - $right[v]$: the pointer to $v$’s right subtree, and
   - $l-size[v]$: the number of nodes in $v$’s left subtree.

   Keys are required to be distinct.

   Do all of the following:

   (a) (20% credit) Write pseudocode for a function $size(r)$ that returns the number of nodes in BST $r$ (passed by reference). Your algorithm should run in time $O(d)$ where $d$ is the depth of the tree.

   (b) (30% credit) Write pseudocode for a function $select(r, i)$ that returns the $i$th smallest key in BST $r$. Your function should assume as a precondition that $1 \leq i \leq s$, where $s$ is the number of nodes in the tree.

   (c) (50% credit) Write pseudocode for a function $insert(r, x)$ that attempts to insert a key value $x$ into BST $r$ at the standard place. The insertion fails iff $x$ is already in the tree. Your function should return a Boolean value, with $true$ indicating that the insertion was successful. Your function should also update $l-size$ attributes of nodes as necessary.

   All three functions should run in time $O(d)$, where $d$ is the depth of the tree. The tree may be empty, in which case $r$ is a null reference. You may use recursion or not as you wish.

3. Given an undirected graph $G = (V, E)$, a vertex cover for $G$ is any set $C \subseteq V$ of vertices such that every edge in $E$ has at least one endpoint in $C$. A vertex half-cover for $G$ is any set $C \subseteq V$ of vertices such that at least half ($|E|/2$) of the edges in $E$ have at least one endpoint in $C$. Recall that the following problem is NP-complete:

   **VERTEX COVER (VC)**
   
   Instance: An undirected graph $G = (V, E)$ and a nonnegative integer $k$.
   
   Question: Is there a vertex cover for $G$ of size at most $k$?

   Consider the following decision problem:

   **VERTEX HALF-COVER (VHC)**
   
   Instance: An undirected graph $G = (V, E)$ and a nonnegative integer $k$.
   
   Question: Is there a vertex half-cover for $G$ of size at most $k$?

   Show that VHC is NP-hard by giving an explicit polynomial reduction from VC to VHC. You are not required to prove that your reduction actually works, but if it doesn’t, some explanation may be worth partial credit. (VHC is clearly in NP, so you are actually showing that VHC is NP-complete.)
Compilers

1. A simple generalized list is defined recursively to be a pair of matching parentheses surrounding a comma-separated sequence of zero or more simple generalized lists. For example,

\[
(())
\]

\[
((((),()(),())())
\]

are all simple generalized lists.

Write a grammar for simple generalized lists. Your grammar should be unambiguous and suitable for LR (bottom-up) parsing. [Hint: two nonterminals and three productions suffice.]

2. Consider the following standard LR (bottom-up) grammar for arithmetic expressions with constants, addition, and multiplication (\(S\) is the start symbol):

\[
S ::= E
\]
\[
E ::= E_1 + T
\]
\[
E ::= T
\]
\[
T ::= T_1 * F
\]
\[
T ::= F
\]
\[
F ::= c
\]
\[
F ::= (E)
\]

(Subscripts have been added to distinguish different occurrences of the same nonterminal in the same production.) Do either but not both of the two items below:

(a) (For 80% credit) Add semantic rules to the grammar above that, given an input expression, produce an equivalent expression with the minimum number of parentheses. So the rules in effect remove unnecessary parentheses.

Your resulting expression should be passed as a string attribute to \(S.\text{output}\). Assume that the terminal \(c\) has a \text{text} attribute that contains the string representing the constant. You may use ‘+’ in your actions to denote string concatenation, and please surround string constants with double quotes.

You should assume that + and * are associative operators, and that the usual precedence rules apply (* before +). Do not rearrange or alter the expression in any way other than the parentheses.

(b) (For 100% credit) Do the same as the last item, but with the grammar also having a production for subtraction:

\[
E ::= E_1 - T
\]

Subtraction has the same precedence as addition, and both operators associate from left to right. Subtraction is not associative.

Examples:
2 + (3 + 4) | 2 + 3 + 4 | addition is associative
(2 * 3) * 4 | 2 * 3 * 4 | multiplication is associative
2 + (3 * 4) | 2 + 3 * 4 | multiplication presides over addition
(2 + 3) * 4 | (2 + 3) * 4 | parentheses needed
(2 − 3) + 4 | 2 − 3 + 4 | operators associate left to right anyway
2 − (3 + 4) | 2 − (3 + 4) | parentheses needed

3. The following fragment of 3-address code was produced by a nonoptimizing compiler:

1 start: i = 1
2 loop1: if i > n goto part2
3 j = 1
4 sum = 0
5 loop2: if j > i goto fin2
6 o = i * 8
7 o = o + j
8 s = a[o]
9 t = j * 8
10 v = a[t]
11 y = s * v
12 if s < n goto fin1
13 sum = sum + y
14 j = j + 1
15 goto loop1
16 fin1: j = sum
17 goto fin2
18 fin2: i = i + 1
19 o = i * 8
20 a[o] = sum
21 goto loop2
22 part2: no-op

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

(a) (20% credit) Decompose the code into basic blocks B₁, B₂, . . . , giving a range of line numbers for each.
(b) (30% credit) Draw the control flow graph, describe any unreachable code, and coalesce any nodes if possible.
(c) (30% credit) Is the variable o live just before line 9? Is the variable y live just before line 14? Explain. Assume that n and sum are the only live variables immediately after line 22.
(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).