Spring 2011 CSE Qualifying Exam
Core Subjects

February 26, 2011

Architecture

1. Consider a memory system that has the following characteristics:

memory	type	line size	miss rate	write policy	% dirty	access time	bandwidth to lower level
L1	split	I: 64 bytes D: 16 bytes	I: 2% D: 7%	D: write-back		0	6.3 GB/s
L2	unified	64 bytes	2%	write back	50%	10 ns	3.2 GB/s
L3	unified	2 KB	1%	write back	60%	50 ns	1.6 GB/s
RAM						400 ns	

   Compute the average memory access time for both instruction fetches and memory references.

2. An (m, n) correlating branch predictor uses the behavior of the most recent m executed branches to choose from 2^m predictors, each of which is an n-bit predictor. A two-level local predictor works in a similar fashion, but only keeps track of the past behavior of itself to predict future behavior. There is a design trade-off involved with such predictors: correlating predictors require little memory for history which allows them to maintain 2-bit predictors for a large number of individual branches (reducing the probability of branch instructions reusing the same predictor), while local predictors require substantial memory for history and are thus limited to tracking a relatively small number of branch instructions.

   For this exercise, assume a $(1, 2)$ correlating predictor that can track four branches (requiring 16 bits) vs. a $(1, 2)$ local predictor that can track two branches using the same amount of memory.

   For the following table of branch outcomes, provide each prediction along with the final mis-prediction rate of each predictor. Assume each predictor is initialized to all taken.

Branch PC (word address)	Outcome
454	T
543	NT
777	NT
543	NT
777	NT
454	T
777	NT
454	T
543	T

3. In this exercise you will show how Tomasulo's algorithm performs when running a common vector loop. The loop is the so-called DAXPY loop (double precision aX plus

Y) and is the central operation in Gaussian elimination. The following code implements the operation $Y = aX + Y$ for a vector of length 100. Initially, R1 is set to the base address of array X and R2 is set to the base address of Y.

```
        DADDIU   R4,R1,#800     ; R1 = upper bound for X
foo:    L.D      F2,0(R1)       ; (F2) = X(i)
        MUL.D    F4,F2,F0       ; (F4) = a*X(i)
        L.D      F6,0(R2)       ; (F6) = Y(i)
        ADD.D    F6,F4,F6       ; (F6) = a*X(i) + Y(i)
        S.D      F6,0(R2)       ; Y(i) = a*X(i) + Y(i)
        DADDIU   R1,R1,#8       ; increment X index
        DADDIU   R2,R2,#8       ; increment Y index
        DSLTU    R3,R1,R4       ; test: continue loop?
        BNEZ     R3,foo         ; loop if needed
```

The functional units are described in the table below:

FU type	Cycles in EX	Number of FUs	Number of reservation stations
Integer	1	1	5
FP adder	4	2	3
FP multiplier	15	2	2

Assume the following:

- Functional units are not pipelined
- There is no forwarding between functional units; results are communicated by the CDB
- The execution stage (EX) does both the effective address calculation and the memory access for loads and stores. Thus the pipeline is IF / ID / IS / EX / WB
- Loads take 1 clock cycle
- The issue (IS) and write result (WB) stages each take 1 clock cycle
- There are 5 load buffer slots and 5 store buffers slots
- Assume that the BNEZ instruction takes 1 clock cycle

For this problem use the single-issue Tomasulo MIPS pipeline with the pipeline latencies from the table above. Show the number of stall cycles for each instruction and what clock cycle each instruction begins execution (i.e., enters its first EX cycle) for three iterations of the loop. How many cycles does each loop iteration take? Report your answer in the form of a table with the following column headers:

- Iteration (loop iteration number)
- Instruction
- Issues (cycle when instruction issues)
- Executes (cycle when instruction executes)
- Memory access (cycle when memory is accessed)
- Write CDB (cycle when result is written to the CDB)
- Comment (description of any event on which the instruction is waiting)

Compilers

1. Provide Semantic actions for the **DoAlt3** control structure,

$$S \rightarrow \textbf{DoAlt3}\ L_1\ \textbf{or}\ L_2\ \textbf{or}\ L_3\ \textbf{until}\ B\ ;$$

   the semantics of which is:

   - execute L_1 then evaluate B,
   - if B evaluates to false then execute L_2 and evaluate B again,
   - if B evaluates to false then execute L_3 and evaluate B again,
   - if B evaluates to false then execute L_1 and evaluate B again,
   - $\vdots$
   - exit the loop when B evaluates to true.

2. Consider the following grammar:

$$\begin{array}{rcl} S & \rightarrow & V\ =\ E \\ E & \rightarrow & E\ *\ E\ \mid\ E\ +\ E\ \mid\ (\ E\ ) \\ E & \rightarrow & V \\ V & \rightarrow & \langle \mathit{Elist} \rangle\ ] \\ V & \rightarrow & \langle \mathit{id} \rangle \\ \langle \mathit{Elist} \rangle & \rightarrow & \langle \mathit{Elist} \rangle\ \mathbf{c}\ E \\ \langle \mathit{Elist} \rangle & \rightarrow & \langle \mathit{id} \rangle\ [\ E \end{array}$$

   (The **c** stands for "comma.")

   (a) Construct the sets of LR(1) items for this grammar. (Stop at 8 sets.)
      i. Compute I_0
      ii. For which symbols x is GOTO(I_0, x) nontrivial?
      iii. For each such symbol x, compute GOTO(I_0, x).
      iv. Compute GOTO$([P, \$], =)$, where P is $S \rightarrow V\ \cdot\ =\ E$.

   (b) For the sets of LR(1) items from above, provide entries of the LR(1) parse table.

3. One of the design decisions for the C programming language is to have arrays start with index zero. This decision was made for the sake of efficiency of the generated code.

   (a) Explain how you can generate code that at run time will compute the address of $a[i_1] \cdots [i_p]$ just as efficiently for a p-dimensional array of the form

$$a[\mathit{low}_1 \ldots \mathit{high}_1] \cdots [\mathit{low}_p \ldots \mathit{high}_p]$$

as it would for such an array where $low_1 = \cdots = low_p = 0$. Here, "just as efficiently" means the same number of additions and multiplications computed at run time. You can assume that the elements of the array have size s and that the array starts at base address b. [Hint: If you don't see the general idea, try writing down the address calculation in both cases for a small value of p (say, $p = 1$ or $p = 2$).]

(b) Provide semantic actions for generating quadruples or 3-address code for two-dimensional array references (two dimensions only). State all assumptions about attributes and contents of the symbol table. For simplicity, assume the indices start with 0.

$$E \rightarrow \langle id \rangle \; [\; E_1 \;] \; [\; E_2 \;]$$

Algorithms

1. (Lopsided Master Theorem) Fix real numbers $0 < \alpha < 1$ and $0 < \beta < 1$. Let $t > 0$ be the unique positive real number such that $\alpha^t + \beta^t = 1$. Suppose $T(n)$ satisfies

$$T(n) = T(\lfloor \alpha n \rfloor) + T(\lfloor \beta n \rfloor) + n^s,$$

for some constant s such that $0 < s < t$. Prove using the substitution method that $T(n) = O(n^t)$. You only need prove the induction step, and you may assume that the floors in the expression are of no consequence. (As usual, $T(n)$ is some positive valued function of n.) [Hint: Observing that $\alpha^s + \beta^s > 1$, prove that $T(n) \leq cn^t - dn^s$ for some positive constants c and d.]

2. Describe an algorithm that takes as input a directed acyclic graph $G = (V, E)$ and a source vertex $s \in V$, and for each vertex $v \in V$ reachable from s,

   - sets the attribute $v.d$ to the *maximum* length of any unweighted path from s to v and
   - sets the attribute $v.\pi$ to be v's predecessor along such a path.

   If $v = s$, then set $v.\pi$ to NIL, and if v is not reachable from s, then set $v.d := -1$ and $v.\pi :=$ NIL. Your algorithm must run in linear time, i.e., $O(|V| + |E|)$. [Hint: Topologically sort G, then use dynamic programming. You may use topological sort without describing how it is implemented.]

3. Consider the following decision problem:

   e-HC
   Instance: An undirected graph $G = (V, E)$ and a distinguished edge $e \in E$.
   Question: Is there a Hamiltonian circuit in G that uses edge e?

   Prove that e-HC is NP-hard by giving a polynomial reduction (aka ptime mapping reduction, aka Karp reduction) from HAMILTONIAN CIRCUIT to e-HC.

Theory

1. (Fenner, Spring 2011) Let $L \subseteq \Sigma^*$ be any language over a finite alphabet Σ. Define $\mathrm{CP}(L)$ (the "common prefix set" of L) to be the set of all $x \in \Sigma^*$ such that x is a prefix of infinitely many strings in L. Show that if L is regular, then $\mathrm{CP}(L)$ is also regular. [Hint: Given a DFA for L, you can build a DFA for $\mathrm{CP}(L)$ with the same number of states.]

2. (Fenner, Spring 2011) Let $\Sigma = \{0, 1\}$. Let $A \subseteq \Sigma^*$ be some nonempty language. Prove that A is decidable if and only if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that

   - f is *length-monotone*, i.e., $|x| \leq |y|$ implies $|f(x)| \leq |f(y)|$ for all $x, y \in \Sigma^*$ and
   - f has range A.

   [For the "if" part, you may want to consider two cases: A finite and A infinite.]

3. (Fenner, Spring 2011) Fix a finite alphabet Σ. We say that a function $f : \Sigma^* \to \Sigma^*$ is *honest* if there exists a polynomial p such that, for all $x \in \Sigma^*$, $|x| \leq p(|f(x)|)$. Informally, f cannot map a really long string to a really short string.

   (a) Prove that if f is polynomial-time computable and honest, then $\mathrm{range}(f) \in \mathrm{NP}$.

   (b) Describe an polynomial-time computable, honest function f whose range is SAT.