Spring 2013 CSE Qualifying Exam
Core Subjects

March 23, 2013

Architecture

1. ROB trace - single issue

   (a) Non-pipelined functional units
   Assume the following:

   - **Nine ROB slots.**
   - 1 integer Execution unit, 5 reservation stations, one cycle to execute.
   - 1 Floating Add Unit, 3 reservation stations, 6 cycles to execute.
   - 1 Floating MULT Unit, 2 reservation stations, 10 cycles to execute.
   - **Functional units are not pipelined.**
   - There is no forwarding between functional units; results are communicated by the common data bus (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 require one clock cycle.
   - The issue (IS) and write-back (WB) result stages each require one clock cycle.
   - There are five load buffer slots and five store buffer slots.
   - Assume that the Branch on Not Equal to Zero (BNEZ) instruction requires one clock cycle.

```
loop:  LD     F0, 0(R1)
       MULT.D  F8, F0, F0
       ADD.D  F2, F2, F8
       DADDIU R1, R1, +8
       BNE    R1, R2, loop
```

   Fill in the table below for as much as you can assuming the BNE is taken and succeeds. (Do not add rows to the table.)

Iter.	Instruction	Issues at	Execute/memory	Write CDB	Commit

(b) ROB - functional units pipelined:
Now assuming that the floating point units are pipelined and that when an operation moves into the second stage of the pipeline the reservation station is freed up.

```
loop:  LD     F0, 0(R1)
       MULT.D  F8, F0, F0
       ADD.D  F2, F2, F8
       DADDIU R1, R1, +8
       BNE    R1, R2, loop
```

Fill in the table below for as much as you can assuming the BNE is taken and succeeds. (Do not add rows to the table.)

Iter.	Instruction	Issues at	Execute/memory	Write CDB	Commit

2. Memory stall cycles Consider a 1GHz computer with the following memory characteristics.

   (a) Cache hit rate is 90%.

   (b) Each cache block contains 32 bytes.

   (c) The processor gets 1 word from its cache with one cycle if it is a cache hit.

   (d) The processor references the memory 1.5 per instruction.

   (e) 25% of the references are writes.

   (f) The bus reads or writes a single word at a time with 100ns.

   (g) The cache uses write allocate on a write miss.

   (h) Assume for the perfect memory, the CPI = 1.0.

   For each of the following two cache write strategies,

   (a) The cache is write through

   (b) The cache is write back

   What is the average CPI? Be sure to state any other assumptions you make clearly.

3. Eliminating stalls:
   Given the code and assume a FOP to FOP stall of two cycles and a Load to FOP stall of one cycle. Also assume FOP to Store has two stalls.

```
$L46:
    addu $2, $2, $3
    l.d $f0, 0($2)
    mul.d $f2, $f0, $f0
    add.d $f4, $f2, $f0
    s.d $f4, 0($4)
    subui $4, $4, #8
    beq $2, $4, $L46
```

   (a) Loop Unrolling SimpleScalar - Show how to unroll the loop just once and schedule to eliminate as many stalls as possible.

   (b) If your loop would normally execute an odd number of times and your unroll it once, what problems arise and how do you address them.

   (c) What in the architecture limits the number of times a loop could be unrolled?

   (d) Show how to unroll this loop for a SuperScalar (VLIW) that can issue one integer, one floating add, one floating multiply, and one load/store operation each cycle.

   (e) How does a fine-grained multithreaded processor such as the Niagara T1 eliminate stalls due to level-one cache misses?

Compilers

1. 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:

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

   (a) (20% credit) Provide productions for $\langle \mathit{StatementList} \rangle$ and $\langle \mathit{Statement} \rangle$. 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.

2. Consider the following two code templates for the while loop:

   (a)
   ```
   execute [[while E do C]] =
           JUMP h
       g:  execute C
       h:  evaluate E
           JUMPIF(1) g
   ```

   (b)
   ```
   execute [[while E do C]] =
       g:  evaluate E
           JUMPIF(0) h
           execute C
           JUMP g
       h:
   ```

   The command `evaluate E` will place the result of evaluating the condition E (with result 1 for true and 0 for false) in such a way that the following `JUMPIF` can access the result.

   (a) Argue that the two code templates are semantically equivalent. Show the object codes produced by the two templates for the source code

   ```
   while i > 0 do i := i - 2
   ```

   (b) Compare the two object codes and explain why most compilers use (a variation of) the second template.

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

```
 1  start:  x := 1
 2          y := 1
 3          sum := x
 4          sum := sum + 1
 5  loop1:  if x = n then goto out1
 6  loop2:  if y = x then goto out2
 7          t1 := a[y]
 8          t2 := y * t1
 9          t3 := t1 + x
10          if t3 < n then goto skip
11          t3 := t3 - n
12  skip:   sum := sum + t3
13          y := y + 1
14          goto loop2
15          goto loop2
16  out2:   a[x] := sum
17          sum := x + 1
18          x := x + 1
19          goto loop1
20  out1:   x := x - 1
21          t1 := a[x]
22          print t1
23          if x = 0 then goto out
24          goto out1
25  out:
```

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 $B_1, B_2, \ldots$, 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 a 25-row table listing which variables are live at which control points. Treat the array `a` as a single variable. Assume that `n` and `sum` are the only live variables immediately before line 25. Your table should look like this:

Before line	**Live variables**
1	...
2	...
3	...
...	...

(d) (20% credit) Describe any simplifying tranformations 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).

Algorithms

1. ...

(a) (75%) Rows and columns of the table below are labeled by statements about functions $f(n)$ and $g(n)$. Fill out all empty cells in this table according to the following rules.

In the cell in a row labeled A and a column labeled B, write a letter **E**, **C**, or **I** denoting

E: the statements A and B are *equivalent*, that is, A implies B and, vice versa, B implies A;

C: the statements A and B are *compatible* but not equivalent, that is, A and B *may* hold simultaneously for certain functions $f(n)$ and $g(n)$, but in general one of the statements does not imply the other;

I: the statements A and B are *incompatible*, that is, A and B *never* hold simultaneously.

X	$g(n) = O(f(n))$	$g(n) = \Omega(f(n))$	$g(n) = \Theta(f(n))$	$g(n) = o(f(n))$	$g(n) = \omega(f(n))$
$f(n) = O(g(n))$					
$f(n) = \Omega(g(n))$					
$f(n) = \Theta(g(n))$					
$f(n) = o(g(n))$					
$f(n) = \omega(g(n))$					

(b) (25%) Prove three of your answers (of your choice): one **E**, one **C**, and one **I**.

2. Given integers $L \leq U$ and a subroutine P computing a non-decreasing function, i.e.,

$$\text{for any integers } a \leq b, \qquad P(a) \leq P(b).$$

The algorithm below returns an integer k in the range $L \leq k \leq U$ such that $P(k) = 0$. If there is no such integer, it returns $L - 1$.

BINARY-SEARCH(L, U)

(a) **if** $P(L) > 0$ or $P(U) < 0$
(b) **return** $L - 1$
(c) $a = L$
(d) $b = U$
(e) **while** $b - a > 1$
(f) $c = \lfloor \frac{a+b}{2} \rfloor$
(g) **if** $P(c) \leq 0$
(h) $a = c$
(i) **else** $b = c$
(j) **if** $P(a) = 0$
(k) **return** a
(l) **if** $P(b) = 0$
(m) **return** b
(n) **return** $L - 1$

(a) (40%) Give an appropriate loop invariant and prove correctness of BINARY-SEARCH algorithm.

(b) (50%) Assuming that computing $P(m)$ takes $f(m)$ time, where f is some non-decreasing positive function, find tight asymptotic lower and upper bounds (in terms of L, U, and f) for the BINARY-SEARCH running time.

(c) (10%) In particular, what is the asymptotic running time in the case of $f(m) = \Theta(1)$?

(Hint: Figure out why this algorithm is called *binary* search.)

3. Given strings U, V, and T, the problem asks to determine whether T includes interweaving (without interruptions) copies of U and V.

   For example, the strings $U = acab$ and $V = ccb$ appear interweaving in $T = b\underline{ac}\bar{c}\underline{a}\overline{cb}\underline{b}d$.

   (a) (30%) Describe the optimal substructure of a solution. Derive and prove a corresponding recurrent formula.

   (b) (50%) Give a pseudocode for a dynamic programming algorithm.

   (c) (20%) Analyze space and time requirements of your algorithm in terms of the lengths of strings U, V, and T.

Theory

1. Prove that the following language is decidable:

$$\{\langle D\rangle \mid D \text{ is a DFA and there exists a string } x \text{ such that } x^n \in L(D) \text{ for all } n \geq 1\}$$

2. Prove that the following language is undecidable:

$$\{\langle G_1, G_2\rangle \mid G_1, G_2 \text{ are CFGs and } L(G_1)L(G_2) = L(G_1)\}$$

3. Let B be a set of TMs with the input alphabet Σ. We define languages

$$A_B = \{\langle M, w\rangle \mid M \in B,\ w \in \Sigma^*, \text{ and } M \text{ accepts } w\}$$

and

$$E_B = \{\langle M\rangle \mid M \in B \text{ and } L(M) = \emptyset\}.$$

(a) (50%) Prove that if $A_B \in NP$, then $E_B \in coNP$.

(b) (50%) Prove that if $A_B \in NP$ and $E_B \in NP$, then $B \in NP$.