Spring 2010 CSE Qualifying Exam
Core Subjects

March 6, 2010
1. Your company has just bought a new dual-core processor, and you have been tasked with optimizing your software for this processor. You will run two applications on this dual core processor, but the resource requirements are not equal. The first application needs 75% of the resources, and the other only 25% of the resources.

(a) Given that 60% of the first application is parallelizable, how much speedup would you achieve with that application if run in isolation?

(b) Given that 95% of the second application is parallelizable, how much speedup would this application observe if run in isolation?

(c) Given that 60% of the first application is parallelizable, how much overall system speedup would you observe if you parallelized it, but not the second application?

(d) How much overall system speedup would you achieve if you parallelized both applications, given the information in parts (a) and (b)?

2. (a) What would be the baseline performance (in cycles, per loop iteration) of the code sequence in Figure 1 if no new instructions execution could be initiated until the previous instructions execution had completed? Use the instruction latencies shown in Figure 2. Ignore front-end fetch and decode. Assume for now that execution does not stall for lack of the next instruction, but only one instruction/cycle can be issued. Assume the branch is taken, and that there is a one cycle branch delay slot.

(b) Think about what latency numbers really mean—they indicate the number of cycles a given function requires to produce its output, nothing more. If the overall pipeline stalls for the latency cycles of each functional unit, then you are at least guaranteed that any pair of back-to-back instructions (a “producer” followed by a “consumer”) will execute correctly. But not all instruction pairs have a producer/consumer relationship. Sometimes two adjacent instructions have nothing to do with each other. How many cycles would the loop body in the code sequence in Figure 1 require if the pipeline detected true data dependencies and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? Show the code with ⟨stall⟩ inserted where necessary to accommodate stated latencies. (Hint: An instruction with latency “+2” needs 2 ⟨stall⟩ cycles to be inserted into the code sequence. Think of it this way: a 1-cycle instruction has latency 1 + 0, meaning zero extra wait states. So latency 1 + 1 implies 1 stall cycle; latency 1 + N has N extra stall cycles.)

(c) Consider a multiple-issue design. Suppose you have two execution pipelines, each capable of beginning execution of one instruction per cycle, and enough fetch/decode bandwidth in the front end so that it will not stall your execution. Assume results can be immediately forwarded from one execution unit to another,
Loop: LD F2,0(Rx)
I0: DIVD F8,F2,F0
I1: MULTD F2,F6,F2
I2: LD F4,0(Ry)
I3: ADDD F4,F0,F4
I4: ADDD F10,F8,F2
I5: ADDI Rx,Rx,8
I6: ADDI Ry,Ry,8
I7: SD F4,0(Ry)
I8: SUB R20,R4,Rx
I9: BNZ R20,Loop

Figure 1.

<table>
<thead>
<tr>
<th>Latencies beyond single cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory LD</td>
</tr>
<tr>
<td>Memory SD</td>
</tr>
<tr>
<td>Integer ADD, SUB</td>
</tr>
<tr>
<td>Branches</td>
</tr>
<tr>
<td>ADDD</td>
</tr>
<tr>
<td>MULTD</td>
</tr>
<tr>
<td>DIVD</td>
</tr>
</tbody>
</table>

Figure 2.

or to itself. Further assume that the only reason an execution pipeline would stall is to observe a true data dependency. Now how many cycles does the loop require?

(d) In the multiple-issue design of part c, you may have recognized some subtle issues. Even though the two pipelines have the exact same instruction repertoire, they are neither identical nor interchangeable, because there is an implicit ordering between them that must reflect the ordering of the instructions in the original program. If instruction $N + 1$ begins execution in Execution Pipe 1 at the same time that instruction $N$ begins in Pipe 0, and $N + 1$ happens to require a shorter execution latency than $N$, then $N + 1$ will complete before $N$ (even though program ordering would have implied otherwise). Recite at least two reasons why that could be hazardous and will require special considerations in the microarchitecture. Give an example of two instructions from the code in Figure 1 that demonstrate this hazard.

(e) Reorder the instructions to improve performance of the code in Figure 1. Assume the two-pipe machine in part c, and that the out-of-order completion issues of part d have been dealt with successfully. Just worry about observing true data dependencies and functional unit latencies for now. How many cycles does your reordered code take?
3. Sorting large datasets is an exciting benchmark because it exercises a computer system across all its components, including disk, memory, and processors. When you do not have enough memory to store the entire dataset into memory, you must sort the data in multiple passes. One common approach is to sort each chunk of the input file and write it to disk; leaving (input file size)/(memory size) sorted files on disk. Then, you have to merge each sorted temporary file into a final sorted output. This is called a “two-pass” sort. More passes are needed in the unlikely case that you cannot merge all the streams in the second pass.

Getting good disk performance often requires amortization of overhead. The idea is simple: if you must incur an overhead of some kind, do as much useful work as possible after paying the cost, and hence reduce its impact. This idea is quite general and can be applied to many areas of computer systems; with disks, it arises with the seek and rotational costs (overheads) that you must incur before transferring data. You can amortize an expensive seek and rotation by transferring a large amount of data.

Your goal is to amortize seek and rotational costs during the second pass of a two-pass sort. Assume that when the second pass begins, there are \( N \) sorted runs on the disk, each of a size that fits within main memory. Our task here is to read in a chunk from each sorted run and merge the results into a final sorted output. Note that a read from one run will incur a seek and rotation, as it is very likely that the last read was from a different run.

(a) Assume that you have a disk that can transfer at 10 MB/sec, with a full cost of 24 ms, and the disk takes 4 ms to complete a full revolution. Assume further that every time you read from a run, you read 1 KB of data, and that there are 100 runs each of size 1 GB. Also assume that writes (to the final sorted output) take place in large 1 GB chunks. How long will the merge phase take, assuming I/O is the dominant (i.e., only) cost?

(b) Now assume that you change the read size from 1 KB to 1 MB. How is the total time to perform the second pass of the sort affected?

(c) In both cases, assume that what we wish to maximize is disk efficiency. We compute disk efficiency as the ratio of the time spent transferring data over the total time spent accessing the disk. What is the disk efficiency in each of the scenarios mentioned above?
Compilers

1. Suppose we want to build expressions out of VAR and CONST tokens with the following allowed operators (eight in all):

<table>
<thead>
<tr>
<th>Operators</th>
<th>Type</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ˆ, $</td>
<td>unary prefix</td>
<td>right to left</td>
</tr>
<tr>
<td>@, #</td>
<td>binary infix</td>
<td>left to right</td>
</tr>
<tr>
<td>*, &amp;</td>
<td>unary postfix</td>
<td>left to right</td>
</tr>
<tr>
<td>%, :</td>
<td>binary infix</td>
<td>right to left</td>
</tr>
</tbody>
</table>

Groups of operators have the same type, precedence, and associativity. Groups are listed in order of decreasing precedence. Expressions can also include parentheses to coerce evaluation order as usual.

Give the rules section of a yacc (bison) source file to build an LALR parser for these expressions. Use expr as the start symbol. Your grammar should reflect the precedence and associativity of the operators as closely as possible. No semantic actions are required. [Your yacc syntax need not be exactly correct as long as it is close.]

2. Consider the following BNF grammar for statements in a modified fragment of the C language:

\[
\begin{align*}
\langle \text{start} \rangle & ::= \langle \text{statement} \rangle \\
\langle \text{statement} \rangle & ::= \langle \text{expr} \rangle ; \\
& \quad | \langle \text{if (expr)} \langle \text{statement}\rangle_1 \text{ else } \langle \text{statement}\rangle_2 \rangle \\
& \quad | \langle \text{while (expr)} \langle \text{statement}\rangle_1 \rangle \\
& \quad | \langle \text{stmtlist} \rangle \\
& \quad | \langle \text{break intconst} \rangle \\
\langle \text{stmtlist} \rangle & ::= */ \text{null derive} */ \\
& \quad | \langle \text{stmtlist}\rangle_1 \langle \text{statement}\rangle
\end{align*}
\]

Here, the start symbol is \langle start \rangle. The intended meaning of the break \text{n} statement (where \text{n} is an integer constant) is to break out of \text{n} enclosing while loops at once. For example,

\[
\{ \\
\quad \text{while (1) } \{ \\
\quad \quad \text{while (1) } \\
\quad \quad \quad \text{if (x) break 1; } /* \text{goes to x=3, same as usual break */} \\
\quad \quad \quad \text{else break 2; } /* \text{goes to x=4 */} \\
\quad \quad \quad \text{x = 3;} \\
\quad \} \\
\}
\]
If \( n < 0 \) or \( n \) is larger than the number of enclosing \texttt{while} loops, then this is an error. (If \( n = 0 \), then the statement has no effect (a no-op); if \( n = 1 \), then it behaves like a normal break statement would.)

Add semantic actions to the grammar above to handle the generalized break statement (you need not worry about anything else). You may define any attributes you find useful. Assume that attribute \texttt{intconst}.val is the numerical value of the \texttt{intconst} token. You may assume the predefined functions

\begin{itemize}
\item \texttt{getNewLabel()} returns a fresh label,
\item \texttt{emitLabel(label)} writes the label to the output stream, followed by a colon,
\item \texttt{emitJump(label)} writes an unconditional jump instruction to the given label, and
\item \texttt{error(msg)} signals an error and gives \textit{msg} as an explanation.
\end{itemize}

[Hint: Consider passing a list of labels as an inherited attribute. Make sure to initialize the list properly in the \langle \textit{start} \rangle-production.]

3. Consider the intermediate code below.

\begin{verbatim}
0  sum = 0
1  i = 0
L0: 2  j = 0
L1: 3  t1 = b
     4  t1 = t1 + i * w
     5  t1 = t1 + j * 8
     6  f = array [ t1 ]
     7  if (f > 0) goto L2
     8  goto L3
L2: 9  sum = sum + f
    10  goto L4
L3: 11 sum = sum - 1
L4: 12  j = j + 1
    13  if (j < 32) goto L1
    14  i = i + 1
L5: 15  if (i < 8) goto L0
L6: 16  return
L7: 17  goto L0
\end{verbatim}

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 18-row table listing which variables are live at which control points. Treat array as a single variable. Assume that n and sum are the only live variables immediately before line 16 (the only exit point). Your table should look like this:

<table>
<thead>
<tr>
<th>Before line</th>
<th>Live variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>17</td>
<td>...</td>
</tr>
</tbody>
</table>

(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).
Algorithms

1. Let $\alpha$, $\beta$, and $\gamma$ be fixed positive real constants such that $\alpha^2 + \beta^2 + \gamma^2 < 1$. Let $T(n)$ satisfy the recurrence

$$T(n) = T(\lfloor \alpha n \rfloor) + T(\lfloor \beta n \rfloor) + T(\lfloor \gamma n \rfloor) + n^2.$$ 

Show by the substitution method that $T(n) = O(n^2)$. (This is tight, since obviously $T(n) = \Omega(n^2)$.) [Hint: $\alpha^2 + \beta^2 + \gamma^2 = 1 - \varepsilon$ for some constant $\varepsilon > 0$.]

2. A palindrome is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, civic, racecar, and aibohphobia (fear of palindromes).

Give an efficient dynamic programming algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input string character, your algorithm should return carac. What is the running time of your algorithm? (Note that a subsequence of a string is not necessarily a substring. Characters in the latter must be contiguous, whereas characters in the former need not be.) [Hint: Let the input reside in the array $S[1 \ldots n]$, where each $S[i]$ is a single character. For all $1 \leq i \leq j \leq n$, let $p[i,j]$ hold a longest palindromic subsequence of $S[i \ldots j]$. How do you fill in the table $p$?]

3. For any fixed integer $k$, we have the following decision problem:

**GRAPH $k$-COLORABILITY ($k$-COL)**

Instance: An undirected graph $G = (V, E)$.

Question: Is there a map $c : V \rightarrow \{1, 2, \ldots, k\}$ (a $k$-coloring of $G$) such that for every edge $(u,v) \in E$ it is the case that $c(u) \neq c(v)$?

It is known that 3-COL is NP-complete.

(a) (40%) Show that, for any fixed $k \geq 3$, the $k$-COL problem is NP-hard by giving a reduction from 3-COL to $k$-COL. (Clearly, for each $k \geq 3$, $k$-COL is in NP, hence it is NP-complete.) [Hint: If you can’t see it for general $k$, try reducing 3-COL to 4-COL first, which is easier.]

(b) (60%) Consider the following decision problem:

**GRAPH HALF-COLORABILITY (HCOL)**

Instance: An undirected graph $G = (V, E)$ with $n$ vertices, where $n$ is even.

Question: Is there a map $c : V \rightarrow \{1, 2, \ldots, n/2\}$ such that for every edge $(u,v) \in E$ it is the case that $c(u) \neq c(v)$?

(Note that here, the number of colors is no longer a fixed constant, but depends on the number of vertices in the graph.) Show that HCOL is NP-hard by giving a polynomial reduction from 3-COL to HCOL. (HCOL is clearly in NP, hence it is NP-complete.) [Hint: Can you adapt your solution to part (a)?]
Theory

1. Fix a finite alphabet $\Sigma$. For any language $A$ over $\Sigma$, define $ADD-ON(A)$ to be the set of all possible strings obtained by starting with a string in $A$ and inserting a single symbol from $\Sigma$ somewhere in the string. That is,

$$ADD-ON(A) := \{xay \mid x, y \in \Sigma^* \text{ and } a \in \Sigma \text{ and } xy \in A\}.$$ 

Show that if $A$ is regular, then $ADD-ON(A)$ is regular. [Hint: Given an $n$-state NFA recognizing $A$, you can construct a $2n$-state NFA recognizing $ADD-ON(A)$.

2. Recall that on any input, a Turing machine (TM) may either accept, reject, or loop (run forever). Acceptance and rejection are both halting behaviors.

Fix an alphabet $\Sigma$. Suppose $A, B \subseteq \Sigma^*$ and $A \cap B = \emptyset$. A separator for $A$ and $B$ is any language $C \subseteq \Sigma^*$ such that $A \subseteq C$ and $C \cap B = \emptyset$. We say that $A$ and $B$ are computably inseparable if there is no decidable separator for $A$ and $B$. Show that the two languages

$$A_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts input string } w\}$$

and

$$R_{TM} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ rejects input string } w\}$$

are computably inseparable.

[Hint: Use a diagonal argument. Suppose that a decidable separator $C$ exists for $A_{TM}$ and $R_{TM}$. Describe a TM $D$ that takes some $\langle M \rangle$ as input ($M$ is a TM), tests whether $\langle M, \langle M \rangle \rangle$ is in $C$, then bases its subsequent behavior on the result of that test. Then consider the string $\langle D, \langle D \rangle \rangle$ to obtain a contradiction.]

3. For any language $L$, define the language

$$\exists \cdot L = \{x \mid (\exists y, |y| = |x|) \text{ } xy \in L\}.$$ 

Show that if $L \in \text{NP}$, then $\exists \cdot L \in \text{NP}$. 
