CSE Qualifying Exam, Fall 2024

CSCE 513-Computer Architecture

1. A MIPS processor with a clock rate of 1GHz and a base CPI of 1.0 is executing the below loop:

```
for (int i=0;i<n;i++){
     for (int j=0;j<m;j++)
          res[i]+= mat[i][j]*vec[j];
}
```

If the memory hierarchy for this processor has the following characteristics:

- L1 hit time = 1 ns, L1 Instruction cache miss rate = 1%, L1D miss rate = 6%
- L2 hit time = 2 ns, L2 miss rate = 2%
- Main Memory access time = 400 ns

Which one of the below options leads to a higher performance improvement compared to the base computer system, and by how much?

(a) Increasing the clock rate to 2GHz
(b) Adding a third-level cache (L3) with a hit time of 0 ns and a miss rate of 25%

State any assumptions you make.

2. **Given**: The below latencies between the dependent Floating-Point (FP) and Integer operations in a MIPS processor:

Instruction Producing Results	Instruction Using Results	Latency in clock cycles
FP ALU operation	FP ALU operation	1
FP ALU operation	Store/Load double	1
Store/Load double	FP ALU operation	1
Store/Load double	Store/Load double	2
Load integer	Integer ALU operation	1
Integer ALU operation	Branch	0
Integer ALU operation	Integer ALU operation	0

Sought: If possible, how many times the below loop should be unrolled to achieve a CPI of 0.5? You can use unrolling, register-renaming, and scheduling to speed up the execution. Draw a table to show the scheduled code for the **2-way superscalar processor**.

```
Loop: l.d    $f0,0($s1)      #FP Load
      sub.d  $f4,$f0,$f2     #FP Sub
      add.d  $f6,$f4,$f0     #FP Add
      s.d    $f6,0($s1)      #FP Store
      addi   $s1,s1,+8       #integer add
      bne    $s1,$zero,Loop #Branch
```

State any assumptions you make.

3. *Computer system Alpha* has a memory bandwidth of 4 GB/s and a peak computational throughput of 4 GFlops/s, while computer system Beta has a peak throughput of 3 GFlops/s and a memory bandwidth of 3GB/s. Given the code below:

```
lv       $v1,0($s0)      #Load Vector
lv       $v2,0($s1)      #Load Vector
mulvv.d  $v3,$v1,$v2     #vector-vector Multiplication
addvv.d  $v4,$v2,$v3     #vector-vector add
subvs.d  $v5,$v4,$F0     #vector-scalar subtraction
sv       $v5,0($s3)      #Store Vector
```

Assuming the *Computer System Beta* fits the entire `$v2` vector in on-chip caches, which computer system is faster in executing the above code by how much? Provide your performance calculation results in GFlops/s. State any assumptions you make.

Spring 2024 CSE Qualifying Exam
CSCE 531, Compilers

1. **Predictive (LL(1)) Parsing**

   Consider the following grammar:

   $$\begin{array}{llll} \text{(i)} & S & ::= & R \\ \text{(ii)} & S & ::= & \mathbf{a}\, S\, \mathbf{c} \\ \text{(iii)} & R & ::= & \\ \text{(iv)} & R & ::= & R\, \mathbf{b} R \end{array}$$

   (a) Show that the grammar is ambiguous, by providing two different syntax trees for the same string. Please use a short string!

   The following grammar is equivalent to the first one but is not ambiguous:

   $$\begin{array}{llll} (1) & S & ::= & R \\ (2) & S & ::= & \mathbf{a}\, S\, \mathbf{c} \\ (3) & R & ::= & \\ (4) & R & ::= & \mathbf{b} R \end{array}$$

   (b) Briefly argue that this grammar is equivalent to the first one and is not ambiguous. No proof is needed.

   From this point on, you will use only the grammar in part (b). You need to build an LL(1) parser for the grammar, following a sequence of steps. Since the grammar is very simple, some the steps will require no work or very little work; still, indicate the result of each step, even if just by writing "no change." Clearly label your work according to step letter. If you need to distinguishe terminals from nonterminals in your answer, please underline the terminals.

   (c) Eliminate left-recursion in the grammar.

   (d) Left-factorize.

   (e) Calculate *Nullable* for each nonterminal.

   (f) Calculate *Nullable* for each production.

   (g) Calculate *FIRST* for every nonterminal, by using set equations.

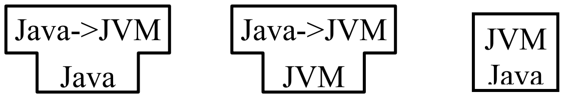
   (h) Calculate *FIRST* for each production. (Hint: are any of the nonterminals nullable?)

   (i) Calculate *FOLLOW* for each nonterminal. (As usual, add a new start production with a new start symbol S' before doing this.)

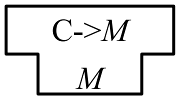
(j) Make an LL(1) parse table for the grammar. Use the following order of nonterminals in the table: S', S, R. Use the following order of terminals in the table: **a**, **b**, **c**, **$**.

(k) Show the input and stack during table-driven parsing of the string **aabbcc$**; use a two-column format, with the left column for the input and the right column for the stack (top element on the left).

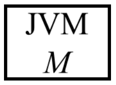
2. **Bootstrapping** Assume that you have a "portable compiler kit" that contains the following components, described using T-diagrams (also known as "tombstone diagram" or "Bratman diagrams"):



(1; 20%; percentages are for guidance only) Suppose that you want to run this kit on some machine M for which there is a C compiler, described by the following tombstone diagram:



Explain how the C compiler is used to obtain the following interpreter:



(2; 10%) Describe the resulting interpretive compiler for machine *M*, using tombstone diagrams to explain how you would compile program P written in Java on machine *M*.

(3; 60%) Bootstrap the interpretive compiler to produce a native compiler whose source is Java and whose target is M, the language of machine *M*: use tombstone diagrams to show each step of your scheme, explaining clearly what needs to be implemented and what can be reused. (Hint: your goal is to avoid writing code in M.)

(4; 10%) There are two common solutions to the previous question. One results in a two-phase compiler, while the other results in a one-phase compiler. If you obtained the two-phase compiler, also describe (in English or by using T-diagrams; your choice) how the one-phase compiler can be obtained. If you obtained the one-phase compiler, also describe (in English or by using T-diagrams; your choice) how the two-phase compiler can be obtained. Briefly describe the tradeoff between the two solutions.

3. **Liveness Analysis and Register Allocation**

Consider the following program.

```
$fib(n)$ 1:  $a := 0$
         2:  $b := 1$
         3:  $z := 0$
         4:  LABEL $loop$
         5:  IF $n{=}z$ THEN $end$ ELSE $body$
         6:  LABEL $body$
         7:  $t := a + b$
         8:  $a := b$
         9:  $b := t$
        10:  $n := n - 1$
        11:  $z := 0$
        12:  GOTO $loop$
        13:  LABEL $end$
        14:  RETURN $a$
```

(a) Compute *succ(i), gen(i), and kill(i)* for each instruction in the program. For your convenience, an example of the table to be filled is provided next to the program.

```
$fib(n)$ 1:  $a := 0$
         2:  $b := 1$
         3:  $z := 0$
         4:  LABEL $loop$
         5:  IF $n{=}z$ THEN $end$ ELSE $body$
         6:  LABEL $body$
         7:  $t := a + b$
         8:  $a := b$
         9:  $b := t$
        10:  $n := n - 1$
        11:  $z := 0$
        12:  GOTO $loop$
        13:  LABEL $end$
        14:  RETURN $a$
```

i	$succ[i]$	$gen[i]$	$kill[i]$
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			

(b) Calculate *in* and *out* for every instruction in the program. Show your work in tabular form. Use of fixed-point iteration is recommended.

(c) Draw the (register-)interference graph for a, b, n, t, and z. Also show the interference table (with columns for statement number, kill set, and "intereferes with" set) that you used to build the interference graph.

(d) Make a four-coloring of the interference graph.

(e) Explain how one could modify the program to use only three registers. You do not need to provide a solution; only describe the approach that you would take.

Fall 2024 CSE Qualifying Exam—Theory (551)

1. Let $\Sigma := \{a, b, c\}$. For any language string $w \in \Sigma^*$, define $\mathrm{MTE}(w)$ to be the set of all strings obtained from w by moving one of its symbols to the end of the string. (MTE stands for "Move To End"). So for example,

$$\begin{aligned} \mathrm{MTE}(abcab) &= \{bcaba, acabb, ababc, abcba, abcab\} \\ \mathrm{MTE}(\varepsilon) &= \emptyset \end{aligned}$$

   (Note that $\mathrm{MTE}(w)$ always includes w because moving its last symbol to the end does not change the string.)

   For any $L \subseteq \Sigma^*$, define
$$\mathrm{MTE}(L) := \bigcup_{w \in L} \mathrm{MTE}(w) \ .$$
   So $\mathrm{MTE}(L)$ is the set of all strings obtained from strings in L by moving any symbol in the string to its end.

   Show by construction that if L is regular, then $\mathrm{MTE}(L)$ is regular. If your construction works, you need not justify it. [Hint: given an n-state DFA for L, there is an NFA for $\mathrm{MTE}(L)$ with roughly $n + |\Sigma|n = 4n$ states.]

2. We assume all languages are over the binary alphabet $\{0, 1\}$ for this problem.

   Let f be a function that, for every enumerator E and natural number $n \geq 0$ as inputs, outputs the number of strings in $L(E)$ of length n, i.e.,
$$f(\langle E, n\rangle) = |L(E) \cap \{0, 1\}^n| \ .$$
   Show that no such f can be computable.

3. The VC-OVERLAP problem is

   Instance: A graph G and a natural number $k \leq |G.V|$.
   Question: Is there a vertex cover C of G of size $\leq k$ such that at least one edge of G has *both* its endpoints of C?

   VC-OVERLAP is clearly in NP. Show that VC-OVERLAP is NP-hard by giving a polynomial reduction to VC-OVERLAP from some well-known NP-complete problem. If your reduction is correct, you need not justify it.

Fall 2024 Qualifying Exam—Algorithms (750)

Question 1 (A Recurrence). Find tight asymptotic bounds on any positive function $T(n)$ satisfying the following recurrence for all sufficiently large n:

$$T(n) = 3T(n/2) + n\sqrt{n}\ .$$

You may assume that any implicit floors and ceilings are of no consequence.

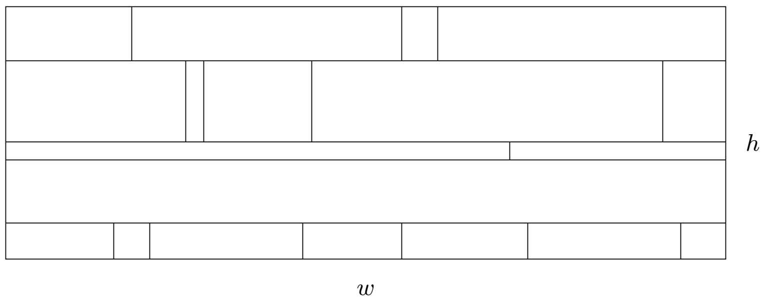
Prove your upper bound by the substitution method. (You do not need to prove the matching lower bound, but if your upper bound is not tight, you will not receive credit even for a correct substitution proof.)

Question 2 (Optimal Restricted Sheet Cutting). You are given a sheet of metal that has width w (the horizontal dimension) and height h (the vertical dimension), where w and h are natural numbers (all units are in centimeters). You can cut the sheet and sell off the pieces. You have a table of prices $P[1..w, 1..h]$ where $P[j,k] \geq 0$ is the amount of money you can charge for a piece of width j and height k. You want to maximize the total amount you can charge selling the pieces.

You are allowed to make the following cuts in order:

- First you can make any number of horizontal cuts through the entire sheet, creating horizontal strips of various integer heights.
- You may then cut each strip into pieces with vertical cuts, independent of how you cut the other strips. The pieces must all have integer dimensions.

For example, here is legal way of cutting the sheet:



Note that cuts are not required. It may be that the best option involves no cuts at all.

Describe an algorithm that finds the maximum total price you can charge from cutting the $w \times h$ sheet. You do not need to find an optimal cut, just its total price. Describe your algorithm in enough detail so that someone who did well in CSCE 750 can implement it without specific knowledge of the problem.

For 95% credit: Your algorithm must run in time $O((wh)^2)$.

For 100% credit: Your algorithm must run in time $O(wh(w + h))$.

You can assume that all numerical arithmetic and comparison operations take $O(1)$ time each. [Hint: dynamic programming.]

Question 3 (Minimum-Weight Directed Cycle). You are given a directed graph G with weight function $w : G.E \to \mathbb{R}$ such that $w(e) \geq 0$ for all $e \in G.E$. You want to find the minimum total weight of any directed cycle passing through a given vertex s. A cycle must include at least one edge (which could be a self-loop).

Describe an algorithm that, given digraph G, weight function w, and $s \in G.V$, returns the minimum total weight of any directed cycle through s, or ∞ if there is no such cycle. You only need to return the weight, not the actual cycle. Describe your algorithm in enough detail so that someone who did well in CSCE 750 can implement it without specific knowledge of the problem.

Your algorithm should run in time $O((V + E) \lg V)$, where $V := |G.V|$ and $E := |G.E|$.