CSE Qualifying Exam, Spring 2023

CSCE 513-Computer Architecture

1. 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	2
FP ALU operation	Store/Load double	2
Store/Load double	FP ALU operation	3
Store/Load double	Store/Load double	3
Load integer	Integer ALU operation	1
Branch	Integer ALU operation	1
Integer ALU operation	Branch	0
Integer ALU operation	Integer ALU operation	0

What is the minimum CPI that can be achieved when executing the below loop on a **2-way out-of-order superscalar processor**? Use unrolling with a *factor of 3* and 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  $f1,$f0,$f2     #FP Sub
      add.d  $f3,$f1,$f1     #FP Add
      s.d    $f3,0($s1)      #FP Store
      addi   $s1,s1,+8       #integer add
      bne    $s1,$zero,Loop #Branch
```

2. Assume a processor that has a clock rate of 2GHz and a base CPI of 1.5 is executing a program with 20% load and stores instructions. If the memory hierarchy for this processor has the following characteristics:

- L1 hit time = 0 ns, L1 Instruction Cache and L1 Data Cache miss rate = 2%
- L2 hit time = 5 ns, L2 miss rate = 5%
- Main Memory access time = 200 ns

(a) What is the effective CPI for this system?

(b) What is the overall speedup achieved if the L2 hit time is reduced to 1.75 ns, and a third-level cache (L3) with a hit time of 2 ns and a miss rate of 1.5% is added to the system?

State any assumptions you make.

3. Assume a computer system with the below characteristics:

- Main Memory bandwidth of 4 GB/s
- A peak computational throughput of 2 GFlops/s
- No cache in the system

(a) What is the expected performance of the system while running the loop below in which the data type of the `res`, `mat`, and `vec` arrays are float (single-precision floating point)?

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

(b) What is the overall speedup achieved if the peak memory bandwidth is increased to 16GB/s and the system is equipped with an on-chip cache that fits the entire `vec` array? State any assumptions you make.

Spring 2023 CSE Qualifying Exam
CSCE 531, Compilers

1. **Syntax-Directed Definition** Consider the following grammar for arithmetic expressions with constants, addition, and multiplication, where S is the start symbol and **c** is a numeric constant:

$$\begin{array}{lcl} E & ::= & E + E \\ E & ::= & E * E \\ E & ::= & \mathbf{c} \\ E & ::= & (E) \end{array}$$

   (a) Show that the grammar is ambiguous.

   (b) Assume that + and * are associative and that, as usual, * has higher precedence that +. Rewrite the grammar to eliminate ambiguity, thus obtaining the standard LR (bottom-up) grammar for arithmetic expressions with constants, addition, and multiplication. (Use subscripts to indicate different occurrences of the same nonterminal in the same production.)

   (c) Write an attribute grammar by adding semanting rules to the grammar you just obtained that, given an input expression, produces 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.output. Assume that the terminal **c** has a text attribute that contains the string representing the constant. Use '+' in your actions to denote string concatenation, and please surround string constants with double quotes. As for the previous part of this question, 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 by removing unnecessary parentheses.

Input	**Output**	**Comment**
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 has precendence over addition
(2 + 3) * 4	(2 + 3) * 4	parenthese needed

2. **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.

3. **Bottom-up (SLR) Parsing** Consider the following grammar:

$$\begin{array}{lcl} A & ::= & aAa \\ A & ::= & bAb \\ A & ::= & \end{array}$$

(a) Describe (in English) the language that the grammar defines.

(b) Is the grammar ambiguous? Justify your answer. Hint: use induction on the length of sentences (in this case, strings) of the grammar.

(c) Construct an SLR parse table for the grammar.

(d) Can the conflicts in the table be eliminated without changing the grammar?

Spring 2023 CSE Qualifying Exam
CSCE 551, Theory

1. Fix an alphabet Σ and let L_1 and L_2 be languages over Σ. Define

$$\text{overlap}(L_1, L_2) := \{xyz : x, y, z \in \Sigma^* \text{ and } xy \in L_1 \text{ and } yz \in L_2\} .$$

So overlap(L_1, L_2) is like concatenation of L_1 with L_2, except that if a string in L_1 has a suffix that equals a prefix of a string in L_2, then the combined string only needs to include that common substring once. For example, if $L_1 = L_2 = \{abb, bba\}$, then

$$\text{overlap}(L_1, L_2) = \{abbabb, abba, abbba, abbbba, bbabb, bbaabb, bbabba\} .$$

Note that $L_1L_2 \subseteq \text{overlap}(L_1, L_2)$, because in the definition, y could just be the empty string.

Show by construction that if L_1 and L_2 are regular, then overlap(L_1, L_2) is regular. If your construction works, you need not justify it.

2. We assume the TM model given in Sipser with a single 1-way infinite tape with cells $0, 1, 2, \ldots$.

Let f be a function such that, for every TM M and string w over M's input alphabet, $f(\langle M, w\rangle)$ outputs a natural number s such that

> *if* M accepts w, *then* it does so using at most s space (which means M's head never scans any cell i with $i \geq s$ during its computation on input w).

Show that no such f can be computable.

(In the definition above, if M loops on input w, then we make no assertions about the value of $f(\langle M, w\rangle)$.)

3. The VERTEX-HALF-COVER problem is

> Instance: A graph G and an integer k.
> Question: Is there a set C of vertices of G such that $|C| \leq k$ and at least half of the edges of G have at least one endpoint in C?

Show that VERTEX-HALF-COVER is NP-complete. To show NP-hardness, polynomially reduce from VERTEX-COVER. You need not prove that your reduction is correct.

Spring 2023 Q-exam — CSCE 750 (Algorithms)

1. **(Solving a Recurrence)** Let $T(n)$ be any positive-valued function defined for all integers $n \geq 0$ by the following recurrence, which holds for all sufficiently large n:

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n \text{ is even,} \\ T(n-1) + n & \text{if } n \text{ is odd.} \end{cases}$$

   Find tight asymptotic bounds on $T(n)$, that is, find a function $f(n)$, as simple as possible, such that $T(n) = \Theta(f(n))$ as $n \to \infty$. Justify your answer using the substitution method.

2. **(Longest Welded Rod)** You are supplied with a sequence $r_1, \ldots, r_n$ of $n > 0$ rods of various positive integer lengths (in inches, say). Your job is to weld (i.e., fused end-to-end) rods to form the longest possible single welded rod. There are two constraints, however:

   (a) The order of the rods cannot be swapped. That is, if $i < j$ and r_i and r_j both appear in the welded rod, then r_i must be somewhere to the left of r_j.

   (b) It may or may not be possible to weld two given rods together.

   Design an algorithm for doing this. Your algorithm takes as input: (1) an array $L[1 \ldots n]$ of positive integers where $L[i]$ is the length (in inches) of rod r_i; (2) an array $W[1 \ldots n, 1 \ldots, n]$ of Booleans, where $W[i,j] =$ TRUE iff it is possible to weld r_i directly with r_j. Your algorithm should return the length of the longest possible welded rod. (You are not required to determine which rods make up the optimal rod.) **Explain** your algorithm well enough so that an intelligent reader (who has taken CSCE 750) with no specialized knowledge can implement it.

   Your algorithm must run in time $O(n^2)$. As usual, you may assume that all arithmetic and comparison operations on integers take $O(1)$ time each.

3. **(Shortest Path)** Dijkstra's algorithm (famously) may fail on a digraph that has negative edge weights. Let $G := (V, E, w)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ that may have *at most one* edge with negative weight. **Design** an algorithm that takes G and two vertices $s, t \in V$ as input and returns the minimum weight of an $s \to t$ path. (Your algorithm is not required to return an actual path, just its weight.) **Describe** your algorithm with enough precision so that an intelligent reader (who has taken CSCE 750) with no specialized knowledge can implement it.

   Your algorithm must run in time $O((n + m) \lg m)$, where $n = |V|$ and $m = |E|$. As usual, you may assume that G is represented by adjacency lists, and all arithmetic and comparison operations on weights take $O(1)$ time each. You may also assume (as usual) that G has no negative-weight cycles. For full credit, **explain briefly** why your algorithm is correct. [Note: The Bellman-Ford algorithm computes shortest paths when weights can be negative, but you cannot simply invoke it because it takes too long to run.]