Spring 2021 CSE Qualifying Exam, Architecture (CSCE 513)

1. A system has a processor with in-order execution that runs at 2.0GHz and has a CPI of 1.2 excluding memory accesses. The only instructions that read or write data from memory are loads (12% of all instructions) and stores (8% of all instructions). The memory system for this computer is composed of a split L1 cache that imposes no penalty on hits. Both instruction cache and data cache are direct mapped. The instruction cache is 64KB with 16-byte blocks, while the data cache is 256KB with 16-byte blocks. The instruction cache has a 5% miss rate. The data cache is write-through and has a 5% miss rate. There is a write buffer on the data cache that eliminates 90% write operation stalls. The 512KB write-back, unified L2 cache is composed with 32-byte blocks and the access time of L2 cache is 2 clock cycles. L2 cache is connected to L1 cache by a data bus which can transfer 128-bit word per clock cycle. Of all memory references sent to the L2 cache, 90% are satisfied without going to main memory. Also, 50% of all blocks in L2 cache replaced are dirty. The main memory has an access latency of 8 clock cycles, after which the transfer rate between L2 cache and main memory is 128-bit word every 2 clock cycles.
   1) What is the average memory access time in nanoseconds for instruction access?
   2) What is the average memory access time in nanoseconds for data read operation?
   3) What is the average memory access time in nanoseconds for data write operation?

2. Consider the following code:

```
for (i = 0; i < m; i++)
  for (j = 0; j < n; j++) {
    a[i] = a[i] + b[j];
    a[j*32 + i] = a[(j+1)*32 + i] * a[(j+2)*32 + i];
  }
```

   1) Is the outer loop of this code parallelable? Please explain your answer.
   2) Is there a potential strategy to execute the inner loop in multiple threads, and how?
   3) Is it possible to unroll the inner loop without knowing the values of m and n, and why?

3. Assuming there is a processor uses Tomasulo's algorithm, incorporating 4-cycle addition/subtraction units, 8-cycle multiplication units and 16-cycle division units. Load or store operation takes just 1 cycle. Considering the following instructions:

```
LD   F2, 8, R1
MULT F0, F2, F4
LD   F3, 12, R1
MULT F1, F3, F4
ADDD F5, F0, F1
SUBD F6, F5, F3
DIVD F7, F6, F0
SD   F5, 0, R1
SD   F6, 4, R1
SD   F7, 8, R1
```

1) Fill the Reservation Stations table at cycle 15.

Reservation Stations (Cycle 15)							
Name	Busy	Op	Vj	Vk	Qj	Qk	Address
Load1							
Load2							
Add1							
Add2							
Add3							
Mult1							
Divd1							
Store1							

2) Fill the Instruction Status table.

Instruction Status			
Instruction	Issue	Execution	Write Result
LD F2, 8, R1	1	2	3
MULT F0, F2, F4	2		
LD F3, 12, R1			
MULT F1, F3, F4			
ADDD F5, F0, F1			
SUBD F6, F5, F3			
DIVD F7, F6, F0			
SD F5, 0, R1			
SD F6, 4, R1			
SD F7, 8, R1			

Spring 2021 CSE Qualifying Exam
CSCE 531, Compilers

1. **Predictive (LL(1)) Parsing.** Consider the grammar below:

Stat	::=	*Stat* ; *Stat*2
Stat	::=	*Stat*2
*Stat*2	::=	*Matched*
*Stat*2	::=	*Unmatched*
Matched	::=	**if** *Exp* **then** *Matched* **else** *Matched*
Matched	::=	**id** := *Exp*
Unmatched	::=	**if** *Exp* **then** *Matched* **else** *Unmatched*
Unmatched	::=	**if** *Exp* **then** *Stat*2
Exp	::=	**id**
Matched	::=	

(Note that the body of the final production is empty.)

(a) Calculate *Nullable* and *FIRST* for every production body in the grammar. (Recall that a string of grammar symbols is *Nullable* if from it one can derive the empty string.)

(b) Calculate *FOLLOW* for every nonterminal in the grammar. (Remember to add the extra start production $S ::= Stat\ \$$.)

(c) Recall that a terminal symbol **a** is in $FOLLOW(N)$ if there is a derivation from the start symbol S of the grammar such that $S \Rightarrow^* \alpha N \beta$ (some authors would write $S \Rightarrow \alpha N \beta$), and $\mathbf{a} \in FIRST(\beta)$. Provide appropriate derivations for each of the symbols in $FOLLOW(Stat)$, $FOLLOW(Stat2)$, $FOLLOW(Matched)$, $FOLLOW(Unmatched)$, and $FOLLOW(Exp)$ that you computed in the previous part of the question.

2. **A Syntax-Directed Definition.** Consider the grammar below for postfix arithmetic expressions built from constants **c** and the binary operators addition (+) and subtraction (−):

$$\begin{array}{lcl} S & ::= & E \\ E & ::= & \mathbf{c} \\ E & ::= & EE+ \\ E & ::= & EE- \end{array}$$

Add semantic rules to the grammar to compute as a string attribute $S.\mathit{infix}$ of the start symbol an equivalent infix expression. The infix expression should have the minimum number of parentheses necessary to ensure a correct result, based on the usual properties of the operators (for example, addition is associative), but you should not assume any commutative law, i.e., you must preserve the left-to-right order of the operands to each operator.

Assume that **c** has an attribute $\mathbf{c}.\mathit{text}$ that holds the actual text of the constant (provided by the lexical analyzer). You may define any other attributes you find helpful. In your rules, surround string constants with double quotes and use + for string concatenation.

For example:

input	$S.\mathit{infix}$
2 3 + 4 +	2 + 3 + 4
2 3 4 + +	2 + 3 + 4
2 3 + 4 −	2 + 3 − 4
2 3 4 − +	2 + 3 − 4
2 3 4 + −	2 − (3 + 4)
2 3 − 4 +	2 − 3 + 4

3. **Object Code Analysis and Optimization** The following fragment of 3-address code was produced by a nonoptimizing compiler:

```
 1  start:  i = 1
 2  loop1:  if i > n goto part2
 3          j = 1
 4          sum = 0
 5  loop2:  if j > i goto fin1
 6          o = i * 8
 7          o = o + j
 8          s = a[o]
 9          t = j * 8
10          v = a[t]
11          y = s * v
12          if s < n goto loop1
13          sum = sum + y
14          j = j + 1
15          goto loop2
16  fin1:   j = sum
17          goto fin2
18  fin2:   i = i + o
19          o = i * 8
20          a[o] = sum
21          goto loop2
22  part2:  no-op
```

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) (30% credit) Draw the control flow graph, describe any unreachable code, and coalesce any nodes if possible.

(c) (30% credit) Is variable `i` live just before line 7? Is the variable `o` live just before line 9? Is the variable `y` live just before line 14? Explain. Assume that `n` and `sum` are the only live variables immediately after line 22.

(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).

Spring 2021 CSE Qualifying Exam
CSCE 551, Theory

1. Let Σ be any alphabet. For any language $L \subseteq \Sigma^*$, define *ADD-ONE*(L) to be the set of all strings obtained from strings $w \in L$ by inserting a single symbol from Σ somewhere into w. Formally,

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

   Show that if L is regular, then *ADD-ONE*(L) is regular. [If your proof involves a correct construction, then you do not need to prove that it is correct.]

2. Let Σ be some alphabet. Let A be a language such that

   - A is Turing-recognizable, and
   - for every $x \in \Sigma^*$, there exists $y \in \Sigma^*$ such that $\langle x, y\rangle \in A$ (although there could be more than one such y).

   Show that there exists a *computable* function $f : \Sigma^* \to \Sigma^*$ such that, for all $w \in \Sigma^*$, $\langle w, f(w)\rangle \in A$. [If your proof involves a correct algorithm for f, then you do not need to prove that it is correct.]

3. Let SUBSET-SUM be the following decision problem:

   > Instance: A nonempty list $\langle a_1, \ldots, a_n\rangle$ of positive integers (in binary) and a positive integer t (in binary).
   > Question: Does there exist a set $J \subseteq \{1, \ldots, n\}$ such that $\sum_{j\in J} a_j = t$?

   It is known that SUBSET-SUM is NP-complete. The closely related PARTITION problem is defined as follows:

   > Instance: A nonempty list $\langle a_1, \ldots, a_n\rangle$ of positive integers (in binary).
   > Question: Does there exist a set $J \subseteq \{1, \ldots, n\}$ such that $\sum_{j\in J} a_j = S/2$, where $S := \sum_{j=1}^{n} a_j$?

   Thus PARTITION is a restriction of SUBSET-SUM where the target is $\frac{1}{2}\sum_{j=1}^{n} a_j$. PARTITION is clearly in NP. Show that PARTITION is NP-complete by giving a polynomial reduction from SUBSET-SUM to PARTITION.

   [If your reduction is correct, you do not need to show that it is correct.]

Spring 2021 Q-exam — CSCE 750 (Algorithms)

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

$$T(n) = 3T(n/3) + 3T(2n/3) + n^3$$

   Find an expression $f(n)$, as simple as possible, such that $T(n) = \Theta(f(n))$. Use the substitution method to **prove** that your answer is correct. (Note: Implicit floors or ceilings in the recurrence do not affect the answer.)

2. **(Connecting wires)** You have a supply of n wires of various positive integer lengths. Each wire has two ends, a *male* end and a *female* end, and each end (either male or female) is one of three types: A, B, or C. You can connect the wires end-to-end (always female-to-male) in any way you please except that ends can only be connected if they have the same type. (Note that a wire's male end may or may not be the same type as its female end.)

   Your goal is find the longest total length of wire you can produce as a single string of connected wires from your supply.

   Describe an algorithm, as efficient as possible, that takes as input three arrays—$L[1..n]$, $M[1..n]$, and $F[1..n]$, where, for all $i \in \{1, \ldots, n\}$, $L[i]$ is the length of the i^{th} wire (a positive integer), and $M[i]$ and $F[i]$ are the types of its male and female ends (either A, B, or C), respectively—and outputs the maximum possible length of a string of wires connected as above. (You only need to find the maximum length; you do not need to output which wires make up a string with that length.)

   Explain how and why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct.

   For full credit, your algorithm must run in time $O(n)$.

3. **(Breadth-First Search)** Maximum credit is 20 points.

   (a) (10 points) Describe the Breadth-First Search (BFS) algorithm, whose input is a simple, undirected graph $G = (V, E)$ (represented by an adjacency list) and a starting vertex $s \in V$. Your description should be detailed enough that a programmer unfamiliar with the procedure (but who is otherwise knowledgable and intelligent) can implement it in time $O(|V| + |E|)$.

   (b) (10 points) A graph is *bipartite* (or 2-*colorable*) if V can be partitioned into two sets V_ℓ and V_r which are both independent, i.e., there are no edges connecting any two vertices in the same set. Describe how BFS can be used to find a V_ℓ and V_r in the case where G is bipartite. For this, you may assume G is connected (in which case, V_ℓ and V_r are unique up to swapping).

   (c) (10 points) It is well-known that G is bipartite if and only if G has no odd-length cycles. How would you augment BFS to output an odd-length cycle in G in the case where G is not bipartite? Again, you may assume that G is connected.