CSE Qualifying Exam, Fall 2023

CSCE 513-Computer Architecture

1. Given: a baseline processor with a clock rate of 1GHz and a base CPI of 1.0 is executing a program with 20% load and stores instructions. The memory hierarchy of this processor has the following characteristics:

- L1 hit time = 2 ns, L1 Instruction Cache and L1 Data Cache miss rate = 1%
- Main Memory access time = 100 ns

Sought: You have a budget of $100 to make some changes to the design to enhance the performance of the computer system. The table below shows the available options and their corresponding costs.

	Item	Cost
1.	Replacing the Main Memory with a new memory with an access time of **75 ns**.	$50
2.	Replacing the CPU with a new CPU with a clock rate of **2GHz** and a CPI of **1.5** for the same program.	$80
3.	Adding a second-level cache with a hit time of **3.5 ns** and a miss rate of **2%**	$30
4.	Replacing the first-level cache with a new cache with a hit time of **1 ns**, and similar miss rates as of the baseline first-level cache	$20

Explain your proposed solution to achieve the best performance possible within the $100 budget. Note that you are allowed to use a combination of the above options as long as you stay within the budget limit.

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	2
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

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 2* 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  $f1,$f1,$f2     #FP Sub
      add.d  $f3,$f1,$f0     #FP Add
      s.d    $f3,0($s1)      #FP Store
      addi   $s1,s1,+8       #integer add
      bne    $s1,$zero,Loop  #Branch
```

3. Given: Computer System Alpha has a main memory bandwidth of 4 GB/s, peak computational throughput of 5 GFLops/s, and no cache.

(a) What is the expected performance of the system while running the loop below?

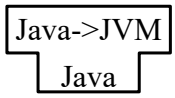
```
Loop: l.s    $f0,0($s1)      #Single Precision FP Load
      l.s    $f1,0($s2)      #Single Precision FP Load
      mul.s  $f2,$f1,$f0     #Single Precision  FP Multiplication
      add.s  $f10,$f2,$f10   #Single Precision FP Add
      s.s    $f10,0($s3)     #Single Precision FP Store
      addi   $s1,s1,+4       #integer add
      addi   $s2,s2,+4       #integer add
      addi   $s3,s3,+4       #integer add
      bne    $s1,$zero,Loop  #Branch
```

(b) Propose two solutions to achieve a **2× speedup** for the computer system while executing the same loop. Your solutions can include any changes to the memory hierarchy or peak computational throughput.

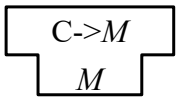
State any assumptions you make.

Fall 2023 CSE Qualifying Exam
CSCE 531, Compilers

1. **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: use tombstone diagrams to show each step of your scheme, explaining clearly what needs to be implemented and what can be reused.

(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.

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

Consider the following ("micro-English") grammar, where terminal symbols and in **bold** and the productions are labeled for convenience. Note the bold period (**.**) at the end of the first production. If you need to distinguishe terminals from nonterminals in your answer, please underline the terminals.

(1)	S	::=	$Sub\ V\ Obj$.
(2a, 2b, 2c)	Sub	::=	**I** | **a** $Noun$ | **the** $Noun$
(3a, 3b, 3c)	Obj	::=	**me** | **a** $Noun$ | **the** $Noun$
(4a, 4b, 4c)	$Noun$	::=	**cat** | **mat** | **rat**
(5a, 5b, 5c)	V	::=	**like** | **is** | **see** | **sees**

You are aksed to build an LL(1) parser for the micro-English grammar by following a sequence of steps. Since the grammar is very simple, some of the steps will require no work or very little work; still, indicate the result of each step, even if just by wrting "no change." Please clearly label your work according to step numbr.

(a) Eliminate left-recursion in the grammar.

(b) Left-factorize.

(c) Calculate *Nullable* for each nonterminal.

(d) Calculate *Nullable* for each production.

(e) Caclulate *FIRST* for every nonterminal, by using set equations.

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

(g) Calculate *FOLLOW* for each nonterminal. (As usual, add a new start production before doing this.)

(h) Make an LL(1) parse table for the micro-English grammar. Use the following order of nonterminals in the table: S', S, Sub, Obj, Noun, V. Use the following order of terminals in the table: **I**, **a**, **the**, **me**, **cat**, **mat**, **rat**, **like**, **is**, **see**, **sees**, **.**, **$**.

(i) Show the input and stack during table-driven parsing of the string **I see a cat.**.

Fall 2023 Q-exam — CSCE 750 (Algorithms)

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

$$T(n) = 2T(n/2) + 16T(\sqrt{n}) + n \ .$$

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

2. **(Maximizing Disjoint Subarrays)** Given an array $A[1 \ldots n]$ of positive integers, we will say that a *k-cluster* (for $1 \le k \le n$) is any subarray of A of length k, that is, $A[i \cdots (i + k - 1)]$ for some i such that $1 \le i \le n - k + 1$. Given an integer $\ell \ge 0$, you want to find up to ℓ many k-clusters in A that maximizes the sum of all the elements in the clusters. Clusters cannot overlap, so each entry of A can belong to at most one k-cluster.

   Design an algorithm that takes as input an array $A[1 \ldots n]$ as above and integers k, ℓ (where $1 \le k \le n$ and $\ell \ge 0$) and returns the maximum total sum of ℓ or fewer non-overlapping k-clusters. You do not need to return an optimal arrangement of clusters, just its sum. **Describe** your algorithm with enough detail so that an intelligent reader (who has taken CSCE 750) can implement it.

   For up to 50% credit, your algorithm must be correct and run in time $O(k\ell n)$. For full credit, your algorithm must be correct and run in time $O(\ell n)$. As usual, you may assume that all arithmetic, comparison, and assignment operations on integers take $O(1)$ time each.

3. **(Shortest Alternating Path)** Let $G := (V, E, w, c)$ be a directed graph with nonnegative edge weight function $w : E \to \mathbb{Z}$ and edge color function $c : E \to \{\textbf{red}, \textbf{blue}\}$. That is, each edge e has weight $w(e) \ge 0$ and is colored either **red** or **blue**.

   Design an algorithm that takes G and vertices $s, t \in V$ as input and returns the minimum weight of any *alternating* s–t path, i.e., a path where each successive edge along the path is colored oppositely from the previous edge. (The first edge, if it exists, can be either color.) If no alternating path exists, the algorithm returns ∞. **Describe** your algorithm with enough detail so that an intelligent reader (who has taken CSCE 750) can implement it. You do not need to output an actual minimum-length alternating path; only the length.

   The worst-case runtime of your algorithm must be $O((|V| + |E|) \lg |V|)$. As usual, you may assume that G is represented by adjacency lists.