Algorithms

Answer each of Problems 1, 2, and 3.

Problem 1: There is a set of n employees $E_1, \ldots, E_n$, and a set of tasks $T_1, \ldots, T_n$. Each task T_i has a difficulty level d_i, and each employee has a skill level s_i. Each employee must be assigned to one task, and every task must be assigned to an employee. All assignments are possible. The goal is to match employees to tasks such that the average difference between each employee's skill level and his or her task's difficulty level is minimized.

In other words, if employee E_i is matched to task $T_{f(i)}$, we wish to minimize

$$\frac{1}{n}\sum_{i=1}^{n} |s_i - d_{f(i)}|.$$

Part (a) (20%): Consider the algorithm that begins with with employee E_1, and assigns that employee the task with the difficulty level that is closest to E_1's skill level. Remove E_1 and the selected task from the pool, and assign employee E_2 the task with difficulty level closest to E_2's skill level. Repeat until all employees and tasks are matched. Give a counterexample showing when this method fails to find the optimal solution.

Part (b) (20%): Describe an $O(n \log n)$ algorithm to find the optimal solution.

Problem 2 (30%): Suppose G is an undirected graph. TERMINOLOGY. The *degree* of a node in G is equal to its number of neighbors. A *simple path* in G is a path in G with no repeated nodes. YOUR TASK. Suppose every node in G has degree at least d. *Prove* that G contains a simple path of length d.

Problem 3 (30%): You have a collection of n biased coins $c_1, \ldots, c_n$, where each coin c_i has a probability p_i of coming up heads when it is flipped. We flip all n coins independently, and want to determine the probability that *exactly* k coins come up heads. Present an $O(n^2)$-time dynamic-programming algorithm to calculate this probability. Assume that we can add or multiply two numbers in $[0, 1]$ in $O(1)$-time.

Architecture/Computer Organization

1. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is 32-bit integer, and A and B are 16×16 integer arrays. *(Hint: A 2D array in C stores in row first order).*

```
for (I = 0; I < 15; I++)
  for (J = 0; J < 15; J++)
    A[I][J] = B[J][I] + A[J][0];
```

   (a) How many 32-bit integers can be stored in a 16-byte cache block?

   (b) References to which variable(s) exhibit temporal locality?

   (c) References to which variable(s) exhibit spatial locality?

   (d) Assume that starting address of matrix A is 0x1000, and the starting address of matrix B is 0x1800. Consider a 512Byte 2-way set associative data *cache with write allocate*. Assume each cache block is 16 Byte. Calculate the cache hit ratio.

   (e) Assume that write back policy with write allocation is used for the data cache. Also assume that the data cache hit time is 1ns, the main memory read and write bandwidth are both 0.5GB/sec. The main memory access latency for both read and write is 100ns. Calculate the average data memory access time.

2. Consider the following loop:

```
        ADD    $t3,   $zero,  $zero   #I0
LOOP:   ADDI   $t3,   $t3,    1       #I1
        ADD    $t1,   $t1,    $t2     #I2
        LW     $t1,   0($t1)          #I3
        LW     $t2,   0($t2)          #I4
        BEQ    $t1,   $t2,    LOOP    #I5
        ADD    $t1,   $t2,    $t3     #I6
        SW     $t1,   0($t2)          #I7
```

   Assume that the branch is taken for the first 100 iterations, *then* it is not taken.

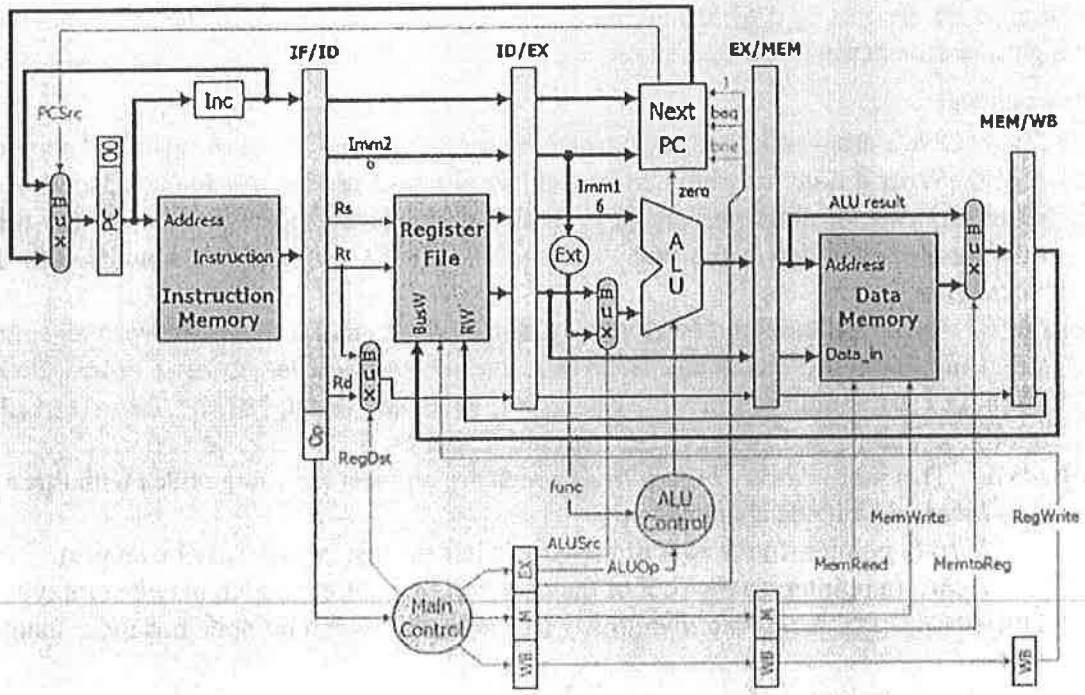
   (a) Consider a MIPS processor with 5-stage pipeline as shown in the figure located on the next page. Assume that the NextPC module is in the EX stage. Also assume that the processor has hazard detection, but no data forwarding and it does not support either delay slot or branch prediction. Complete the pipeline execution diagram on the supplied ***Architecture/Computer-Organization Answer Sheet*** to show how each instruction is processed during the first 16 cycles. Mark the ID of the corresponding instruction in the first column of the diagram. For this problem, assume that the branch is always taken.

   (b) How long does it take to complete the above instruction sequence (considering that the branch is taken for the first 100 times and then it is not taken)?

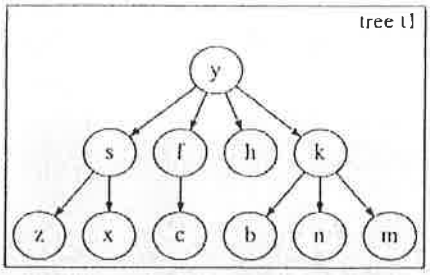
(c) For all the multiplexers in the following figure, a "0" on the selector connects the output to the upper input, while a "1" on the selector connects the output to the lower input. During the execution of the instruction sequence, at which clock cycles will the control signal PCSrc be set to 1? How about ALUSrc?

(d) After moving the NextPC module to the ID stage and adding data forwarding function to the processor, how much performance improvement in terms of cycle reductions can be obtained for the above instruction sequence?

(e) Modify the instruction sequence so that it works correctly on a processor with 1 delay slot.



Data Structures & Programming



BACKGROUND: This problem concerns *multiway trees* with character labels such as the ones pictured to the right. Note that the subtrees of a node are ordered left-to-right. To represent such multiway trees we use nodes with two fields:

label: a Unicode character

subtrees: the list of the subtrees of this node in left to right order (i.e., the list starts with the leftmost subtree and continues left-to-right).

Provide C, C++, Haskell, Java, or ML code to answer the following questions. Clarity counts—if your answers are too hard to follow, they will be marked wrong. All the questions have short, straightforward answers.

QUESTIONS:

(a) *(5%)* Give a data-structure for representing multiway-tree nodes as described above.

(b) *(20%)* Write a function `postorder` that, given a tree, returns a string consisting of the post-order sequence of character values in the tree. The function should run in $O(n)$-time where n is number of nodes in the tree. EXAMPLE: For `t1` the function would return the string `"zxscfbnmky"`.

(c) *(30%)* Write a function `level` that, given a tree `t` and a non-negative integer `k`, returns the string of all the characters at level `k`, preserving their left-to-right order. EXAMPLES: `level(t1,0)` should return `"y"`, `level(t1,1)` should return `"sfhk"`, `level(t1,2)` should return `"zxcbnm"`, and `level(t1,3)` should return `""`.

(d) *(45%)* This part involves *binary trees* which are represented using nodes with three fields:

label: a Unicode character

left: (a pointer to) the root of the node's left subtree (which may be empty)

right: (a pointer to) the root of the node's right subtree (which may be empty)

TERMINOLOGY: A *restricted multiway tree* is one in which no node has more than two subtrees.

Write two functions:

- `toBinary`, which takes a multiway tree *mt* (which we assume is restricted) and returns *mt*'s translation to an equivalent binary tree. (When a multiway node has only one subtree, make this subtree the left subtree in the translation.)
- `toMulti`, which takes a binary tree, *bt*, and returns a translation of this to an equivalent multiway tree.

For this problem, you should:

(i) *(5%)* Give a data-structure for representing binary-tree nodes as described above.

(ii) *(20%)* Make sure that for each multiway tree *mt*: `toMulti(toBinary` *mt*`)` is structurally identical to *mt*.

(iii) *(20%)* Make sure that for each binary tree *bt*: `toBinary(toMulti` *bt*`)` is structurally identical to *bt*.

The tricky one is (iii) because of binary tree nodes that have an empty left-subtree, but nonempty right-subtree. To handle this you are allowed to adjust your representation of multiway trees to allow, say, empty multiway trees which can act as place-holders for "missing" left-subtrees.

Operating Systems

1. (Process, Threads, and Scheduling, 35%)

   (a) Assume the system has only user-level threads. Describe the actions taken by a kernel to context switch between processes.

   (b) Now consider a system with kernel-level threads. What optimizations can the kernel make when context switching between threads of the same process?

   (c) The Sun UltraSparc (this was before the days of multicore processors) had multiple register sets. Describe the actions taken by the kernel if the new context is already loaded into one of the register sets. What must the kernel do if the new context is in memory (in a TCB stored in the kernel thread table), but not one of the register sets, and all the register sets are in use?

   (d) Consider a real-time system that needs to handle three processes: two voice calls that each run every 5 msec and consume 1 msec of CPU time per burst, plus one video at 25 frames/sec, with each frame requiring 20 msec of CPU time. Is this system schedulable? If not, explain why not. If so, prove it by calculating a schedule.

2. (Deadlock, 65%)

   (a) What are the four necessary conditions for deadlock? Explain what each one means.

   (b) What are deadlock prevention, avoidance, and recovery? Explain what each one means.

   (c) For deadlock prevention, describe a scheme to make it impossible for deadlock to occur. Indicate which of the four necessary conditions you are eliminating.

   (d) A student majoring in anthropology and minoring in computer science has embarked on a research project to see if African baboons can be taught about deadlocks. He locates a deep canyon and fastens a rope across it, so the baboons can cross hand-over-hand. Several baboons can cross at the same time, provided that they are all going in the same direction. If eastward-moving and westward-moving baboons ever get onto the rope at the same time, a deadlock will result (the baboons will get stuck in the middle) because it is impossible for one baboon to climb over another one while suspended over the canyon. If a baboon wants to cross the canyon, it must check to see that no other baboon is currently crossing in the opposite direction. Write a program using semaphores that avoids deadlock. Do not worry about a series of eastward-moving baboons holding up westward-moving baboons indefinitely (i.e., you need not worry about starvation). Be sure to indicate whether your semaphores are counting semaphores or mutex semaphores (you may use either or both in your solution). Be explicit about what state, if any, is shared between eastward-moving and westward-moving baboons.

   (e) Repeat the previous problem, but now avoid starvation. When a baboon that wants to cross to the east arrives at the rope and finds baboons crossing to the west, he waits until the rope is empty, but no more westward-moving baboons are allowed to start until at least one baboon has crossed the other way.

Programming Languages

The next page contains the syntax and operational semantics for a simple language of arithmetic expressions.

Your Tasks: (do all three)

1. Modify the semantics so that, in additon to *evaluating* arithmetic expressions, it also *calculates the cost* of evaluation as follows:

   - Each variable lookup costs 2.
   - Each application of an arithmetic operation costs 1.
   - Each variable update (i.e., update of a single variable's value in the state) costs 3.
   - Evaluation of numerical constants is free (i.e., costs 0).

   For example, the cost of evaluating the expression $(\underline{3} + \mathsf{inc}\ y) * (\underline{17} + y)$ is 10 , because it requires two variable lookups (cost of 4), three applications of arithmetic operations (cost of 3), and one variable update (cost of 3).

   Note: There is more than one way to do this, such as modifying configurations or by labeling transition relations. Briefly (but clearly) describe your approach before giving your semantics.

2. Consider adding the following two new constructs to the language:

   - $x := E$

     Intended semantics: Evaluate expression E in the current state and assign that value to variable x (no other state changes occur). The value of expression $x := E$ is the calculated value of E.
   - $\mathsf{clock}\ E$

     Intended semantics: Evaluate expression E to determine the cost associated with its evaluation; that cost is the value of the expression $\mathsf{clock}\ E$. (All state changes that occur during E's evaluation stay in effect.)

   Add the necessary rules for these new constructs to your semantics.

3. Let s_1 be a state in which the variables a, b, and c all have value 1.

   Using your semantics, give a formal derivation for the evaluation of the following expression in the state s_1:

   $$b := (\mathsf{inc}\ a * \underline{7}) + \mathsf{clock}\ \ (c + a)$$

Syntax The syntax relies on the following sets:

- **Num**, the set of *integer numerals*, ranged over by meta-variable $\underline{n}$
- **Var**, the set of *variables*, ranged over by meta-variable x
- **AExp**, the set of *arithmetic expressions*, ranged over by meta-variable E

Based on these sets (and the stated conventions for meta-variables), the abstract syntax for the set **AExp** is defined by the following grammar:

$$E \ ::= \ \underline{n} \mid x \mid \mathsf{inc}\, x \mid E_1 + E_2 \mid E_1 * E_2$$

Semantics For the semantics, we make use of the following fairly standard conventions:

- We use n as a meta-variable ranging over the set $\mathbb{Z}$ of *integer values*.
- Numerals are distinguished from integer values by underlining. For example, $\underline{3}$ is a numeral (i.e., syntax), whereas 3 denotes a semantic value.
- Likewise, *plus* and *mult* denote addition and multiplication over integer values, in contrast to $+$ and $*$ (which are syntax).
- A *state* σ is a finite partial function from variables to integer values:

  $$\sigma : \mathbf{Var} \rightharpoonup \mathbb{Z}.$$

  Thus, for any variable x, $\sigma(x)$ is the value associated with x.
- $\sigma[v/x]$ denotes the state that is identical to σ *except* for mapping x to v.

Operational Semantics ($\Rightarrow$) :

$$\textbf{Num}\ \frac{}{\langle \underline{n}, \sigma\rangle \Rightarrow \langle n, \sigma\rangle} \qquad \textbf{Var}\ \frac{}{\langle x, \sigma\rangle \Rightarrow \langle n, \sigma\rangle}\ \sigma(x) = n$$

$$\textbf{Inc}\ \frac{}{\langle \mathsf{inc}\ x, \sigma\rangle \Rightarrow \langle n, \sigma[(n+1)/x]\rangle}\ \sigma(x) = n$$

$$\textbf{Add}\ \frac{\langle E_1, \sigma\rangle \Rightarrow \langle n_1, \sigma_1\rangle \qquad \langle E_2, \sigma_1\rangle \Rightarrow \langle n_2, \sigma_2\rangle}{\langle E_1 + E_2, \sigma\rangle \Rightarrow \langle n, \sigma_2\rangle}\ n = plus(n_1, n_2)$$

$$\textbf{Mult}\ \frac{\langle E_1, \sigma\rangle \Rightarrow \langle n_1, \sigma_1\rangle \qquad \langle E_2, \sigma_1\rangle \Rightarrow \langle n_2, \sigma_2\rangle}{\langle E_1 * E_2, \sigma\rangle \Rightarrow \langle n, \sigma_2\rangle}\ n = mult(n_1, n_2)$$

Digital Logic Circuit Design

Instructions

For this part of the exam, do your work on the special sheets provided. These sheets include the questions for this part of the exam.

Programmable Logic Unit Design

(a) 1-Bit PLU Tabular Specification

F1	F0	f(A,B)
0	0	B'
0	1	A and B
1	0	A or B
1	1	0

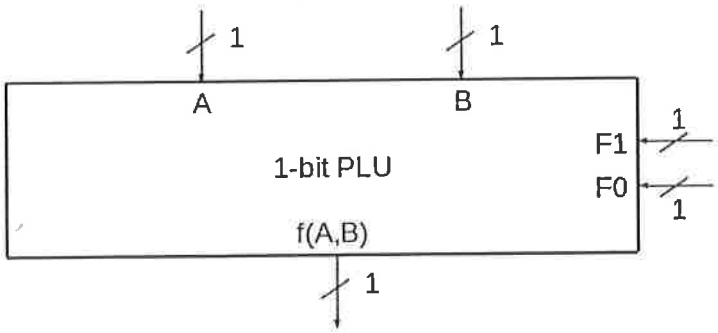


Figure 1: 1-bit Programmable Logic Unit Specification

Figure 1 shows a 1-bit programmable logic unit (PLU), where F1 and F0 specify which logic function f(A,B) to compute. Note: all the inputs and outputs are 1-bit.

Fill in the table below that completely specifies the operation of the 1-bit PLU.

F1	F0	A	B	f(A,B)
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

(b) 1-Bit PLU Logical Specification

Express the PLU's behavior as a logical formula using Shannon's expansion. Specifically, if PLU(F1,F0,A,B) is the PLU's function, then

$$\begin{aligned} PLU(F1,F0,A,B) = & F1' \cdot F0' \cdot A' \cdot PLU(0,0,0,B) + F1' \cdot F0' \cdot A \cdot PLU(0,0,1,B) + \\ & F1' \cdot F0 \cdot A' \cdot PLU(0,1,0,B) + F1' \cdot F0 \cdot A \cdot PLU(0,1,1,B) + \\ & F1 \cdot F0' \cdot A' \cdot PLU(1,0,0,B) + F1 \cdot F0' \cdot A \cdot PLU(1,0,1,B) + \\ & F1 \cdot F0 \cdot A' \cdot PLU(1,1,0,B) + F1 \cdot F0 \cdot A \cdot PLU(1,1,1,B) \end{aligned}$$

Fill in the following table

PLU(0,0,0,B) =

PLU(0,0,1,B) =

PLU(0,1,0,B) =

PLU(0,1,1,B) =

PLU(1,0,0,B) =

PLU(1,0,1,B) =

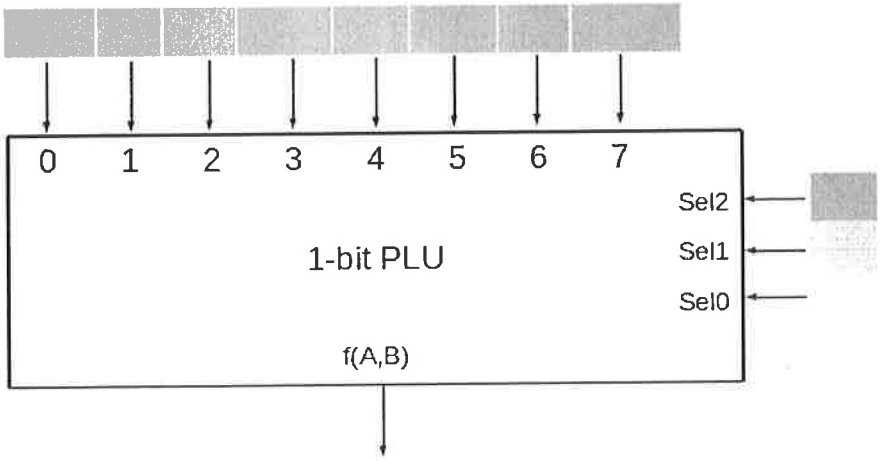
PLU(1,1,0,B) =

PLU(1,1,1,B) =

Based on the above table, write a logical expression for PLU(F1,F0,A,B). Take advantage of the simplifications for terms that are 0 or 1 when using Shannon's expansion.

(c) 1-bit PLU Implementation

Implement the 1-bit PLU using an 8-1 multiplexer



(d) 4-bit PLU Implementation

Draw your implementation of a 4-bit PLU using as many 1-bit PLUs as needed.

Controller Design (Finite-State Machine Design)

Implement the following use-case description using combinational logic and a state register.

Use case description: Create a completely specified deterministic four-state FSM with a single input *x* and a single output *Out* with the following behavior:

1. The initial state is S_0. If the input *x* is 1, the machine transitions to state S_1. If the input *x* is 0, the machine changes to state S_2. The output *Out* is 0 in state S_0.
2. If the machine is in state S_1, the output *Out* is 0. If the input *x* is 1, the machine changes state to S_3. If the input *x* is 0, the machine changes state to S_2.
3. If the machine is in state S_2, the output *Out* is 0. If the input *x* is 1, the machine changes to state S_1. If the input *x* is 0, the machine changes state to S_3.
4. If the machine is in state S_3, the output *Out* is 1. Regardless of the value of input *x*, the machine transitions to S_0.

Fully document each stage of your design by including:

1. A state-transition diagram,
2. A truth table whose inputs are the current machine state and input variables, and whose outputs are the next-state functions and output functions, and
3. Logical formulas for next-state and output functions specifying the combinational logic necessary to implement the machine using two D flip-flops, two 4:1 multiplexers, and one 2:1 multiplexer.
4. The implementation using D flip-flops, two 4:1 multiplexers, and one 2:1 multiplexer.

Use the standard binary coding for your state encoding, i.e.,

$$S_0 : s_1 s_0 = 00, S_1 : s_1 s_0 = 01, S_2 : s_1 s_0 = 10, S_3 : s_1 s_0 = 11$$

(a) State-transition Diagram

Draw the state-transition diagram described by the use case.

(b) Next-state and output truth table

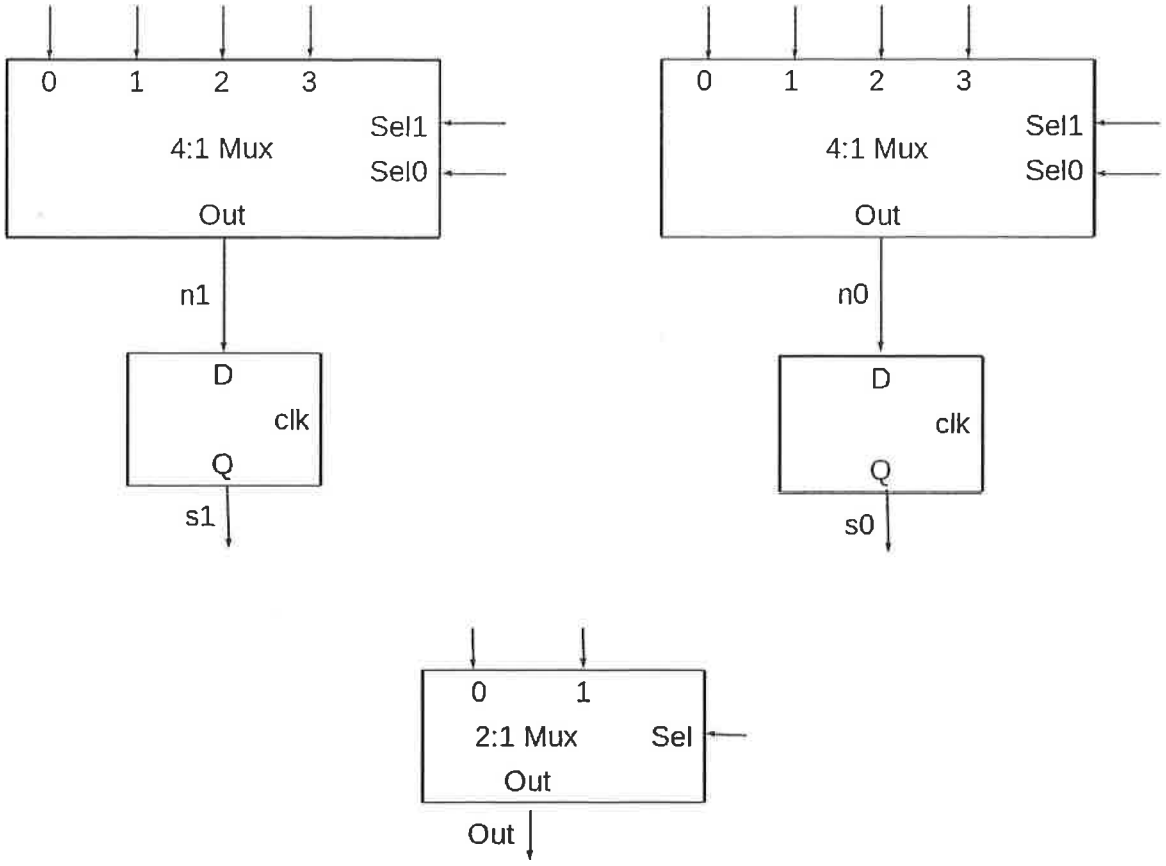
Present State, Input			Next State, Output		
s1	**s0**	**x**	**n1**	**n0**	**Out**
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

(c) Logical formulas for next-state and output functions

Write logical formulas for *n1*, *n0*, and *Out*. Minimization is not necessary.

(d) Implementation

Fill in the inputs below that implement the controller


0
1
2
3
4:1 Mux
Sel1
Sel0
Out
n1
D
clk
Q
s1
0
1
2
3
4:1 Mux
Sel1
Sel0
Out
n0
D
clk
Q
s0
0
1
2:1 Mux
Sel
Out
Out