COMP 273, Fall 2020, Assignment 4

School of Computer Science, McGill University
Available On: Saturday, November 21st, 2020.
Due Date: Saturday, December 5th, 2020 by 11:59 pm.

By handing in your solutions using *myCourses*, you declare that you have followed the assignment submission instructions at the end of this assignment.

Late policy: 10% off of the total marks, per day late, for up to 2 days. If submitted 48 hrs or more after the deadline, your assignment will not be accepted.

1 Memory Mapped I/O - Banking Application (70 marks)

The goal of this question is to create a mini banking application using MMIO in MARS. There are two types of accounts: Checking and Savings. Each bank account is represented by a unique 5-digit number. Each account has a balance. The operations that can be performed include: a) opening an account b) finding out the balance, c) making a deposit, d) making a withdrawal, e) transferring between accounts, f) taking a loan, g) closing an account, h) displaying query history and i) quitting the application.

All I/O operations must be performed using MMIO. Operations are executed when the appropriate commands are input and the <CR> (carriage-return) key is pressed.

5 marks is given for the overall flow of the program. The transactions should build on one another. You can add other variables as you need them in the .data section.

A single 1D array containing words will be used to store all banking information in the following format:

```
[checking acct no,  savings acct no,
checking balance,  savings balance, loan]
```

The contents of this array are to printed to the standard I/O window after each operation. The elements in the array are originally initialized to 0. An example of a populated array is:

```
[12345, 67890, 1500, 670, 200]
```

A second array will be used to hold the last 5 queries performed. The array will always have 5 elements, which are integers, for each query, and older queries can be pushed out of the array. This array should be updated after each operation. The second array can thus be a 25x1 array, where each row corresponds to the banking array's contents after previous transactions.

Table 1 shows the list of operations and the corresponding command format. A subroutine should be created for each operation.

Table 1: List of Banking Application Operations.

S/N	**Operation**	**Command Format**	Subroutine name
1	Open Account (Checking)	CH Acct-No Opening-Balance	open_account
2	Open Account (Savings)	SV Acct-No Opening-Balance	open_account
3	Deposit	DE Acct-No Amount	deposit
4	Withdraw	WT Acct-No Amount	withdraw
5	Loan	LN Amount	get_loan
6	Transfer	TR Acct-No-From Acct-No-To Amount	transfer
7	Close	CL Acct-No	close_account
8	Balance	BL Acct-No	get_balance
9	History	QH Number-of-Queries	history
10	Quit	QT	quit

Note:

1. Each command and the subsequent numbers (account number or balance) are separated by at least a single space. Extra spaces should be ignored.

2. The commands have to be in capital letters (e.g., CL, LN, TR etc.). Small letters are not allowed.

3. An error message should be displayed if an invalid instruction is entered and the user should be prompted to enter a valid command. The error message should always be:

   "That is an invalid banking transaction. Please enter a valid one."

4. The balance, deposit, withdrawal or loan amount will not exceed 10000000 (10 million).

5. The values from MMIO should be converted from ASCII to integers when saving them as words representing integers in the arrays. This makes computations and printing easier. **A smart idea would be to write a subroutine that converts an ASCII string of chars, corresponding to digits 0-9, to the equivalent integer value.**

6. The program should terminate when the user enters the QT.

1.1 Account Opening (10 marks)

The two types of account that can be created are a checking account and a savings account. The commands to create a checking account and savings account are CH and SV followed by the account number and opening balance. This operation can only be performed if there is no existing account of that type. An error message should be displayed otherwise.

For example, if the user enters CH 10101 500, a checking account with account number 10101 and a balance of 500 should be created. The banking information array should have the following values:

```
[10101, 0, 500, 0, 0]
```

If the user then enters *SV* 20202 17500, a savings account with account number 20202 and a balance of 17500 should be created. The updated banking information array becomes:

```
[10101, 20202, 500, 17500, 0]
```

Note that an operation only occurs when the <CR> key is pressed. An error message should be displayed if the command format is not correct and the application should wait for new instructions to be keyed in.

1.2 Operations (35 marks)

The three operations that can be performed are deposit, withdrawal, transfer, loan and close account. All these operations can only take place if the account mentioned actually exists.

Make a Deposit (5 marks)

The deposit operation is triggered when the following command is entered: *DE* 20202 100. An amount of 100 is deposited into the account with account number 20202. The updated banking information array becomes:

```
[10101, 20202, 500, 17600, 0]
```

Make a Withdrawal (10 marks)

The withdrawal operation is triggered when the following command is entered: *WT* 10101 200. An amount of 200 is withdrawn from the account with account number 10101 if the current balance is greater than the withdrawal amount. Else, an error message should be displayed. The updated banking information array becomes:

```
[10101, 20202, 300, 17600, 0]
```

If a withdrawal is made from a savings account, a 5% fee (5% of the withdrawal amount) is deducted from the account. For example, if the following command is entered *WT* 20202 1000, a 50 fee is deducted from the savings account and the updated banking information array becomes:

```
[10101, 20202, 300, 16550, 0]
```

Take a Loan (10 marks)

A loan can be taken from the bank if the account holder has a combined amount of greater than $10,000$ in both accounts. The loan cannot exceed 50% of the total account balance - in such a case an error message is displayed. A loan is triggered if the following command is entered: LN 2000. In this particular case a loan of 2000 is given to the user since it meets the loan requirements. The updated banking information array becomes:

```
[10101, 20202, 300, 16550, 2000]
```

Transfer between Accounts (5 marks)

A transfer can be made from either account to the other. A transfer is triggered when the following command is entered: TR 20202 10101 4000. An amount of 4000 is transferred from the savings account 20202 to the checking account 10101. You have to check that there is enough balance in the originating account, else an error message is displayed. The updated banking information array becomes:

```
[10101, 20202, 4300, 12500, 2000]
```

A loan is payed back using the transfer operation by entering the following command TR 10101 1000. This command has 3 entries, not 4. In this case an amount of 1000 is withdrawn from the checking account to pay 1000 out of the loan. You have to check that there is enough money in the account and the amount being paid back is not more than the loan amount. An error message should be displayed otherwise. The updated banking information array becomes:

```
[10101, 20202, 3300, 12500, 1000]
```

Closing an Account (5 marks)

An account is closed if the following command is entered: CL 10101. If the other account is still open, all the balance in the account to be closed is transferred to the other account. Thus, the updated banking information array becomes:

```
[0, 20202, 0, 15800, 1000]
```

However, if the other account is closed and the following command is entered CL 20202, the remaining balance is used to clear pending loans. Any "extra" money is lost.

Thus, the updated banking information array becomes:

```
[0, 0, 0, 0, 0]
```

NOTE: An error message should be triggered if the format for any of the operations is incorrect.

1.3 Queries (15 marks)

The two queries that can be performed are a balance inquiry and a query history.

Get Account Balance (5 marks)

This operation is triggered when the following command is entered: BL 20202. The current balance in the account with account number 20202 is returned. If the current banking information array is $[10101, 20202, 5000, 300, 50]$, the command BL 20202 will return:

```
$300
```

Query History (10 marks)

This operation should return the last n contents of the bank array up to a maximum of 5. Thus the command $QH4$ will return:

```
          [0, 0, 0, 0, 0]
    [0, 20202, 0, 15800, 1000]
[10101, 20202, 3300, 12500, 1000]
[10101, 20202, 4300, 12500, 2000]
```

If $0 <= n > 5$, an error message should be displayed. Also if n is greater than the number of elements in the query array, an error message should be displayed.

1.4 Quit Program (5 marks)

Your program should terminate only when the user enters QT. This should be the only time when your program terminates.

2 Finite State Machine (30 marks)

Consider a circuit that has a data input *Data* and a reset input *Reset* and a single output Z.

The output Z should be asserted true (set to 1) whenever the string of 1s and 0s entered since the last assertion of *Reset*, when interpreted as an unsigned binary integer, is divisible by 5. Otherwise it should be set to false (0). Assume that the two inputs act independently, i.e., they can change at the same time. Also, approach this problem using a Moore machine, where the output Z's value is associated with being in a particular state, and not with the transition itself. This will be explained further in class - the parity-checker FSM is an example of a Moore machine. Finally, use the following properties:

1. If $a\%5 = \alpha$ and $b\%5 = \beta$ then $(a + b)\%5 = (\alpha + \beta)\%5$.
2. If a current string has a numerical value of x then placing a 0 on the right hand side results in a string having a value $x + x$ and placing a 1 results in a string having a value $x + x + 1$.

Derive a symbolic state transition table for this problem (a finite state machine). Now, choose a state encoding and then, using D flip-flops that are rising-edge triggered, derive expressions for the output Z and the new states as functions of the current states and the current inputs.

Draw the underlying digital circuit using **Logisim-evolution**. There is a single *Data* input and a single *Reset* input and a single output Z. The sampling of inputs should be controlled by a clock, i.e., the inputs are sampled at each rising edge of the clock. Test your circuit with various sequences of inputs to make sure it works as expected.

ASSIGNMENT SUBMISSION INSTRUCTIONS

Each student is to submit his or her own unique solution to these questions, electronically, in *myCourses*. By handing in this assignment, you declare that the work you are submitting is your own.

1. You have been given starter code: *bank.asm* in the Assignment 4 folder. Your solution to the question should assemble and run in MARS. Comment your code as much as possible to make it easy to grade and give partial marks.

2. The logisim circuit(s) must run under logisim-evolution, to be graded. We will assume that you have tested it.

3. Zip your files: your PDF file, assembly file and Logisim circuit into a single file and rename it with your student ID number, e.g., 260763964.zip. Ensure that you use only the *.zip* format and no other compression software e.g., .rar, .7z, etc. Make sure that you submit a single file (the zipped file), not many files.

4. Submit this single compressed file on myCourses under Assignment 4.

5. Hints, suggestions and clarifications may be posted on the discussion board on *myCourses* as questions arise. Even if you don't have any questions, it is a good idea to check the discussion board.

6. Once you have submitted your assignment, download the files you uploaded and check that it is indeed what you intended us to grade. This step is critical because a non-trivial number of you will submit the wrong files, or a corrupted version. You cannot submit a corrected file later, i.e., after the submission deadline and the two day "late" window have passed.