ECEN-304 Spring 2018: Term Project

Machine Instructions Simulator (MIS)

Assigned: Thursday, March 29s

Goals

This is a programming project that is designed to let you utilize all the concepts and apply all the libraries, tools, and advanced programming techniques explained in class to build a middleware for the Machine Instruction Simulator (MIS), which is in the form of a virtual machine capable of executing an instruction set. The project is composed of two phases, where the second phase builds on top of the first one. It is required also to conduct a UML Object Oriented Design and maintain it through constructive updates and evolution through out the development and the buildup of the middleware.

Problem Description

MIS is a virtual machine that is designed to execute a set of instructions. An MIS program is represented by a set of variable declarations and a set of instructions that operates on the set of defined variables. A program is introduced to the MIS in a text file with the extension ".mis" and the results are stored in 2 files; ".out" which should include the output of a the target MIS running program (until the end of the program or the first error), and ".err" which should include all the parse and runtime errors generated by the MIS virtual machine during execution.

An MIS program is built up of two sections. The first section defines all the variables to be used throughout the program execution, and the second section has all the instructions to be executed which builds up the program's sequential logic. For simplicity, all variables in an MIS program are global variables where local variables or subroutines are not supported by the MIS virtual machine. Every variable definition and program instruction needs to be written on a separate line, and should not exceed 1024 characters.

The MIS specification supports 4 data types from which variables can be declared and defined. The following table presents all the MIS variable types:

Name	**Description**	**Size in Bytes**
NUMERIC	Non-Decimal numeric value	8
REAL	Decimal numeric value	8
CHAR	A character	1
STRING	A character array	Range of 1- 256

The following template is used to define variables in an MIS program, and variable names should start with $:

VAR <name>,<type>,<optional: size in case of string only>, <default value>

Examples:

```
VAR $myint,NUMERIC,100
VAR $myfloat,REAL,12.14
VAR $mychar,CHAR,'c'
VAR $mystr,STRING,100,"Hello world MIS!!!"
```

The MIS specification supports 16 instructions as defined in the following table:

Instruction	**Number of Parameters**	**Parameters**	**Description**
ADD	3-13	1: REAL or Numeric Variable Rest: Real or Numeric Variable or Constant.	Adds all parameters excluding the first one and store the results in the first parameter.
SUB	3	1: REAL or Numeric Variable Rest: Real or Numeric Variable or Constant.	Subtract the third parameter from the second parameter and store the result in the first parameter.
MUL	3-13	1: REAL or Numeric Variable Rest: Real or Numeric Variable or Constant.	Multiply all parameters excluding the first one and store the results in the first parameter.
DIV	3	1: REAL or Numeric Variable Rest: Real or Numeric Variable or Constant.	Divide the second parameter by the third parameter and store the result in the first parameter. **Note:** divide by zero should be detected and reported, and the program should tolerate crashes resulting from divide-by-zero exceptions.
ASSIGN	2	1: Any Variable. 2: Variable or constant of the same type as parameter 1.	Store the second parameter into the first parameter.

OUT	1-12	Any variables or constants	Prints out the parameters to the standard output.
SET_STR_CHAR	3	1: String variable 2. Variable or constant representing an index. 3: A variable or constant character.	Set a string character at specific index to a character. **Note:** index range should be checked and errors should be reported.
GET_STR_CHAR	3	1: String variable 2. Variable or constant representing an index. 3: A variable character	Store the character at the index equivalent to parameter 2 into the third parameters.
LABEL	1	1: Inline Label.	Sets a placeholder that a JMP instruction can go to.
JMP	1	1: Label name to jump to.	Change execution sequence by setting the execution pointer to the first instruction after the label.
JMP(Z/NZ)	2	1: Label name to jump to. 2: A numeric variable or constant.	Apply the logic of JMP based on the condition result. Z: second parameter is zero NZ: second parameter is not zero
JMP(GT/LT/GTE /LTE)	3	1: Label name to jump to. 2: A numeric variable or constant. 3: A numeric variable or constant.	Apply the logic of JMP based on the comparison condition result between the seconds and the third parameters. GT: P2 > P3 LT: P2< P3 GTE: P2 >= P3 LTE: P2<= P3
SLEEP	1	1: A numeric variable or constant.	Suspend execution for a number of seconds.

Note: in the above table "numeric" means NUMERIC or REAL.

Parameters of an instruction are separated by commas, and it is very important to highlight that type checking should be performed for all instruction types.

The following are examples of instructions:

```
ADD $myint,100,20.7,300,$myint1
MUL $myint,100,20.7,300,$myint1
SUB $myint,100,$myint2
```

```
DIV $myfloat,100,$myint2
ASSIGN $myint,$myint1
ASSIGN $mystr1,$mystr2
ASSIGN $mystr1,"This is a test string"
SET_STR_CHAR $mystr1,21,'\n'
LABEL LAB1
JMP LAB1
JMPZ LAB1,$myint
JMPGTE LAB1,$myint1,$myint2
SLEEP 100
SLEEP $myint
```

Please refer to the last section, which presents some examples of MIS programs.

Phase 1

In this phase you are required to conduct an object oriented UML design that includes at least use case scenarios, detailed class diagrams, and detailed sequence diagrams.

You are also required to submit a full implementation of the MIS virtual machine that runs sequentially. You should apply all the elements of the object model studied and explained in class, and try to utilize all the C++11/14 features possible, such as STL containers. You are also expected to utilize concepts like inheritance, templates, virtual methods, and polymorphism in general.

Phase 2

In this phase you are required to build a client/server model for your MIS virtual machine. You will be splitting your virtual machines into a client and server programs. The client will be responsible for reading the program from a file and sending it to the server over network sockets where it should execute and the server should return the results to the client where it should be stored in the corresponding output files based on the type of output returned; error or standard output.

Phase 3

in this phase you are required to convert your MIS virtual machine server to be capable of handling more than one client concurrently at a time, which entails that different MIS programs can be executed in parallel by the same server virtual machine, and each one should have its own separate isolated context.

Finally, you are requested to implement multi-threading programmatic constructs within your MIS virtual machine. You should achieve that by introducing **five** new instructions to your MIS virtual machine implementation, which are:

Instruction	**Parameters**	**Description**
THREAD_BEGIN	No Parameters	Designate the beginning of a detached thread
THREAD_END	No Parameters	Designate the end of a detached thread
LOCK	Variable Name	Lock a variable, where the first instruction to lock a variable will pass through while subsequent instructions will block until the preceding instruction is released by UNLOCK. LOCK can be only invoked from within a thread.
UNLOCK	Variable Name	Release an acquired lock on a variable. UNLOCK should be ignored in case it is invoked by a thread that did not acquire the lock in the first place. UNLOCK can be only invoked from within a thread.
BARRIER	No Parameters	The BARRIER instruction can be invoked only from within the main programs, and it will block the main program until all the detached running threads are finished. If initially there is no detached threads the main program should resume and ignore the BARRIER instruction.

In this phase you are required to utilize the POSIX Sockets library, and you have the choice to choose between TCP and UDP, but you will need to present design justification for your choice. You have the choice between using the pthreads library or the built in C++11 muti-threading library that is part of the C++ STD.

Moreover, you will need to present your updated version of you UML design with modification to the original version you submitted in phase 1, if there are any updates.

What to submit

1. UML Design document in PDF format including all UML diagrams and design details.
2. All your code.

3. Your code should be split among header files (.h) and source files (.cpp), and all your implementation need to be in the source files. You need to have a very good reason for including implementation in header files, and an explanation of such motives need to be included in your report.
4. Very detailed in-code documentation.
5. Make file.
6. A report describing how to build the software using g++, including all design decisions taken and their alternatives, and explaining anything unusual to the teaching assistant. The report should be in PDF format.
7. Contributions files; each group member will provide a file named after her/his name explaining what she/he did and her/his contribution in the project.
8. Do not submit any object, or executable files. Any file uploaded and can be reproduced will result in grade deductions.

Important Assumptions:

Some specifications that are built by default in the compiler's builds should not be assumed for granted. Please read the documentation of QNX as an example of a real-time operating system that withdraws some of the features granted by some builds of the compiler for better performance. http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.ide.userguide%2Ftopic%2Fmemory_Illegal_dealloc_.html.

This is just an example, and my point is to urge you to try to be as conservative as possible with practices you usually do and are guarded when the compiler default build options are being enabled. You are developing code that you do not know who is going to be using it, and where is it going to be deployed.

How to submit:

Archive everything, in a continent folder tree format and upload it to Canvas under the corresponding phase; what I mean by "continent folder tree" is to create a folder for related content; e.g. code, design, ...etc.

This is a group project and you should elect a group captain among your group members, and all the submissions will be performed through his/her Canvas account. IMPORTANT: EVERY GROUP SHOULD SUBMIT THROUGH THE GROUP CAPTAIN ONLY.

Deadline:

1. **Phase 1:** Sunday 29th April 2018, until 11:59 pm.
2. **Phase 2:** Tuesday 8th May 2018, until 11:59 pm.
3. **Phase 3:** Sunday 27th May 2018, until 11:59 pm.

Grading

This assignment is worth 30% of the overall course grade. The assignment will be graded on a 100% grade scale, and then will be scaled down to the 30% its worth. The grading of the assignment will be broken down according to the following criteria:

1. **Phase 1:**
    a. UML Design and Final Report 15%
    b. Sequential MIS 30%
    c. Documentation 5%
2. **Phase 2:**
    a. Updated UML Design document if applicable: 5% penalty if the design is changed and an updated UML design is not submitted.
    b. Client/Server sequential MIS 10%
3. **Phase 3:**
    a. Updated UML Design document if applicable: 5% penalty if the design is changed and an updated UML design is not submitted.
    b. Client/Server Multi-threaded MIS 15%
    c. Programmatic constructs for multi-threading. 20%
4. Final Version of the Documentation & Report 5%

The project will be evaluated based on the correct execution of the MIS virtual machine, the code correctness and neatness, design decisions aspects, and how much you have utilized the concepts of abstraction, encapsulation, modularity, and hierarchy. It is very important to highlight that although documentation is worth 5% in each phase, yet missing or unclear documentation might result in more grade deduction if the grader cannot verify the correctness of your code and logic, and the provided documentation is not sufficient to clarify such matters.

Delays

You have up to 3 calendar days of delay in each phase, after which the corresponding phase will not be accepted and your grade in that case will be ZERO. For every day (of the 3 allowed days per phase), a penalty of 5% will be deducted from the total grade.

MIS Example Programs

MIS Program 1:

sequential_hello.mis:

```
VAR $mystring, STRING,100,”Hello world MIS:”
NUMERIC $counter,0

EQUAL $counter,10
LABEL LOOP
        OUT $mystring,$counter,” !!\n”
        SUB $counter,$counter,1
JMPNZ LOOP,$counter
```

Execute the program via the following command:

```
misvm sequential_hello.mis
```

MIS Program 2:

distributed_parallel_hello.mis:

```
VAR $mystring, STRING,100,"Hello world MIS:"
NUMERIC $counter1,0
NUMERIC $counter2,0
NUMERIC $counter3,0
NUMERIC $sleep1,0
NUMERIC $sleep2,0
NUMERIC $sleep3,0

ASSIGN $counter1,10
ASSIGN $counter2,20
ASSIGN $counter3,30
ASSIGN $sleep1,3
ASSIGN $sleep2,2
ASSIGN $sleep3,1

THREAD_BEGIN
        LABEL LOOP1
                OUT "Thread 1: ",$mystring,$counter1," !!\n"
                SUB $counter1,$counter1,1
                SLEEP $sleep1
        JMPNZ LOOP1,$counter1
THREAD_END

THREAD_BEGIN
        LABEL LOOP2
                OUT "Thread 2: ",$mystring,$counter2," !!\n"
                SUB $counter2,$counter2,1
                SLEEP $sleep2
        JMPNZ LOOP1,$counter2
THREAD_END

THREAD_BEGIN
        LABEL LOOP3
                OUT "Thread 3: ",$mystring,$counter3," !!\n"
                SUB $counter3,$counter3,1
                SLEEP $sleep3
        JMPNZ LOOP3,$counter3
THREAD_END

OUT "Forked Threads and will sleep on Barrier\n"
BARRIER
OUT "Out of Barrier and exiting program\n"
```

To execute this program you need your MIS server daemon to be running on the local or a remote node listening on a specific network port, and you should issue the following command on the client side to start execution:

```
client_misvm <server IP address> <server port> distributed_parallel_hello.mis
```