Design Note

1. Part 1

1.1 GUI

The UI should be designed using javaFX, which have less code and better view than swing. The first step is to design the layout of the user interface and it should have some free space to be expend during the part two and part three. It will divide into five parts by using border layout. The top menu will show all buttons, such like “run”, “single step”, “restart”, and “load”. The center content should display all registers and using ratio button to display the binary value of every register. The right content will used to manage the memory and user can operate memory and look up the value in specific address easily. The bottom area will used to design a user console where user check all instructions that have been executed.

1.2 Registers

In our opinion, registers should be a single class. There are 14 attributes whose type are integer in this class. Each of these attributes represent a register’s value such as pc, ir, mar, r1, x1 and so on. Accordingly, there should be 28 methods for us to use to get or set these attributes. Furthermore, we design a method to reset all attributes to zero. In addition, since after executing an instruction the value of pc would plus one, we should design a method called pcPromote to make the value of pc plus one. Moreover, since we may need to find the corresponding register based on instructions which are binary string, we should design methods that use instruction as parameter to get or set the value of specified register.

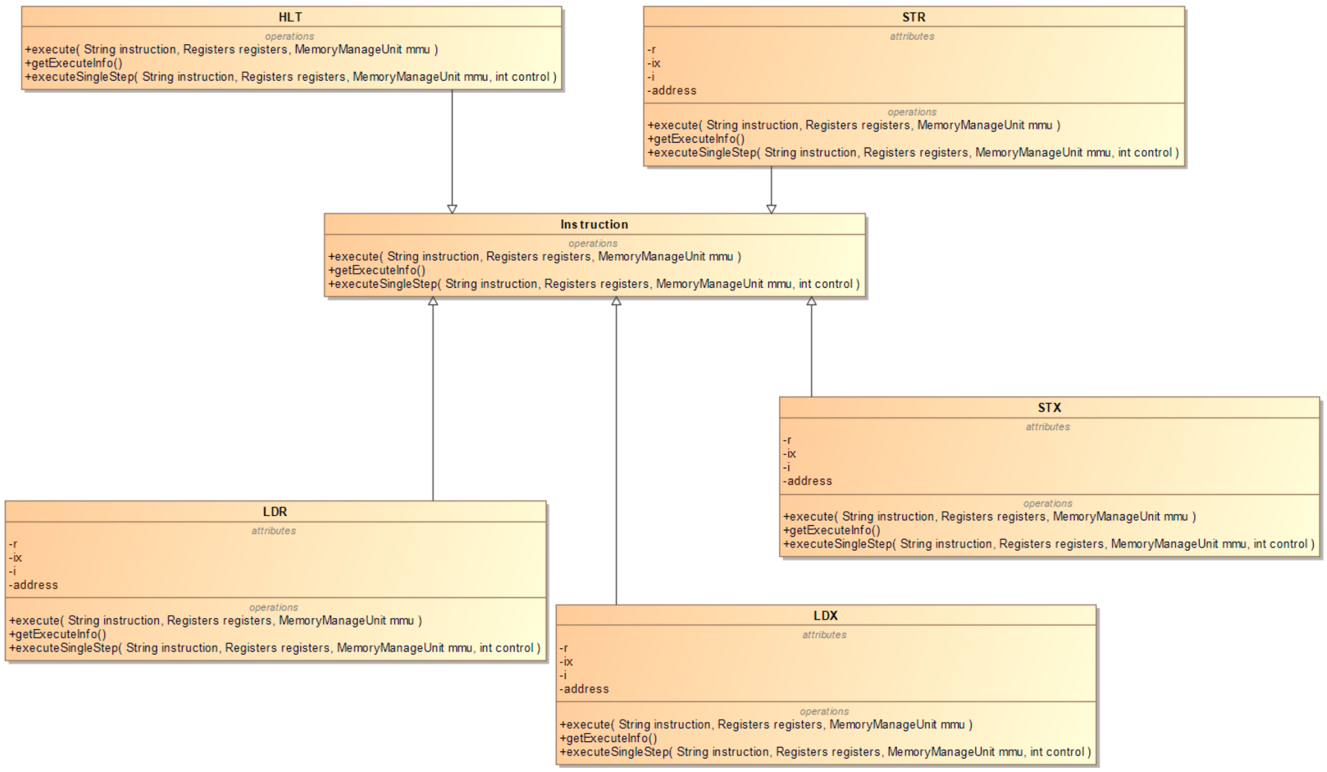
1.3 Memory

Based on the requirement in the project description and the feature of the real memory. We choose List as data structure to implement the memory in the computer. First of all, the address and value must can be accessible by CPU component and it should provide write and read function. Therefore, the List is very ideal to due to it is ordered and it also provide build-in method to get access to the index of the List and also value. Apart from this, the constructor should initialize the List with all value at zero. The memory can be an attribute in the GUI, because we need a front panel to display all address and value in the memory.

1.4 Instructions

In our opinion, there should be a class for each instruction. And execution should be methods in the class. Specifically, in each instruction class, there should be attributes to record each part of the instruction for later use. Then, a method called getExecuteInfo to return the instruction information which will be showed in the user panel. Thirdly, a method called execute which requires instruction (type: String), Registers (type: Object), MemoryManageUnit (type: Object) to execute what this instruction do. Moreover, we also design a method called executeSingleStep to execute the instruction by process, for example, in LDR class, the first time we call executeSingleStep method, the value of MAR will change, then if we call executeSingleStep method again, the value of MBR will change.

1.5 Class Diagram


HLT
operations
+execute(String instruction, Registers registers, MemoryManageUnit mmu)
+getExecuteInfo()
+executeSingleStep(String instruction, Registers registers, MemoryManageUnit mmu, int control)
STR
attributes
-r
-ix
-i
-address
operations
+execute(String instruction, Registers registers, MemoryManageUnit mmu)
+getExecuteInfo()
+executeSingleStep(String instruction, Registers registers, MemoryManageUnit mmu, int control)
Instruction
operations
+execute(String instruction, Registers registers, MemoryManageUnit mmu)
+getExecuteInfo()
+executeSingleStep(String instruction, Registers registers, MemoryManageUnit mmu, int control)
STX
attributes
-r
-ix
-i
-address
operations
+execute(String instruction, Registers registers, MemoryManageUnit mmu)
+getExecuteInfo()
+executeSingleStep(String instruction, Registers registers, MemoryManageUnit mmu, int control)
LDR
attributes
-r
-ix
-i
-address
operations
+execute(String instruction, Registers registers, MemoryManageUnit mmu)
+getExecuteInfo()
+executeSingleStep(String instruction, Registers registers, MemoryManageUnit mmu, int control)
LDX
attributes
-r
-ix
-i
-address
operations
+execute(String instruction, Registers registers, MemoryManageUnit mmu)
+getExecuteInfo()
+executeSingleStep(String instruction, Registers registers, MemoryManageUnit mmu, int control)

2. Part 2

2.1 Cache

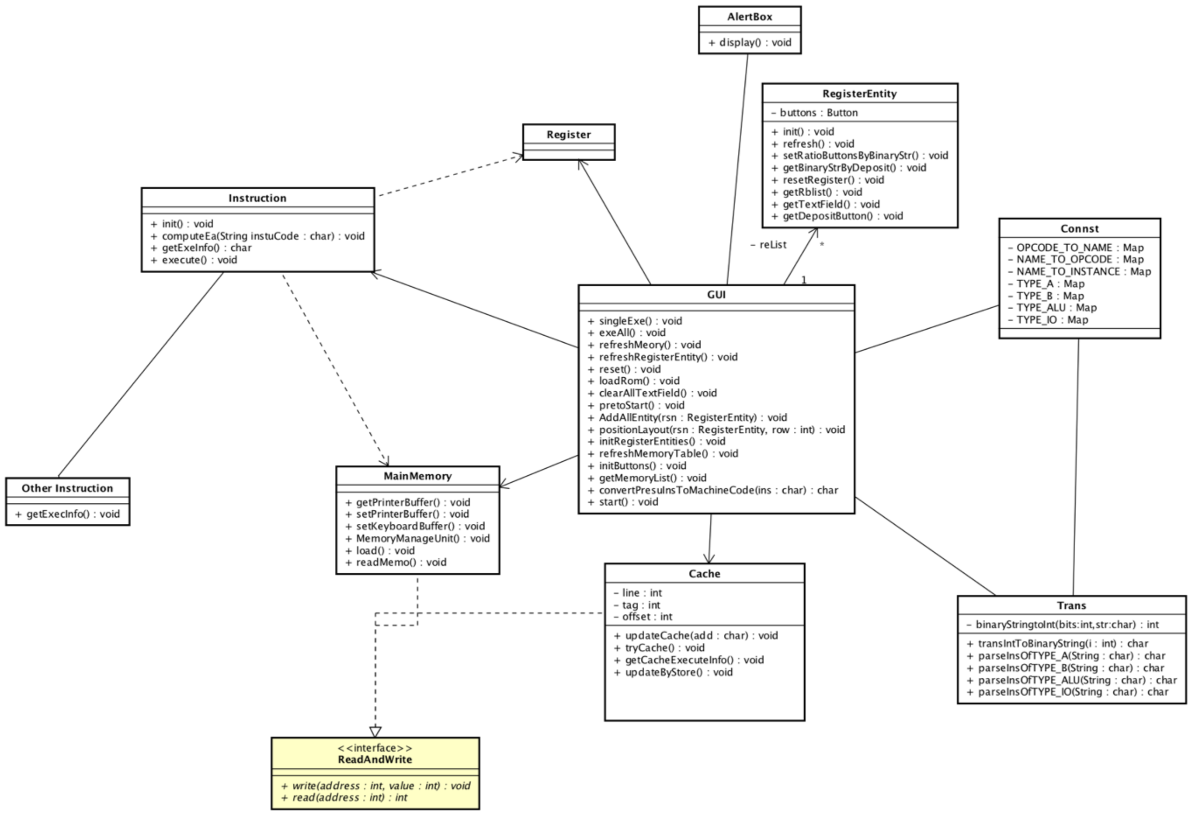
In our opinion, there should be a class for Cache. And try to find desired data or update cache should be methods in the class. A method called tryCache will first divide the address into three parts, then according to the parts return the designated data and show whether it is found in cache. If it is not in the cache, the system will fetch the data from memory, and change the according block in cache. Moreover, a method called update will check if the data being written is in cache or not, if it is in the cache, the according content in the cache will be update as well.

2.2 Compiler

In order to reduce the work of write program 1, we are going to implement a complier which can complier the source code instead of machine code. The solution to realize this function is using the HashMap data structure provided by java. When loading the source code into simulator, the complier can receive all code and convert them into machine code to run the CPU part.

The first step is to consider a way to filter the source code file because it may have some comments. We can use some special character like "//" and "#" to mark the line as comments. The second step is to analysis the source code into machine code. In this part, we need to classify types of code. By observing the format (the needed argument) of every instruction, we classify them into four parts and also need write corresponding method to compile them into machine code represented by zero and one.

2.3 Improved Class Diagram


AlertBox
+ display() : void
RegisterEntity
- buttons : Button
+ init() : void
+ refresh() : void
+ setRatioButtonsByBinaryStr() : void
+ getBinaryStrByDeposit() : void
+ resetRegister() : void
+ getRblist() : void
+ getTextField() : void
+ getDepositButton() : void
Register
Instruction
+ init() : void
+ computeEa(String instuCode : char) : void
+ getExeInfo() : char
+ execute() : void
- reList
*
1
Connst
- OPCODE_TO_NAME : Map
- NAME_TO_OPCODE : Map
- NAME_TO_INSTANCE : Map
- TYPE_A : Map
- TYPE_B : Map
- TYPE_ALU : Map
- TYPE_IO : Map
GUI
+ singleExe() : void
+ exeAll() : void
+ refreshMeory() : void
+ refreshRegisterEntity() : void
+ reset() : void
+ loadRom() : void
+ clearAllTextField() : void
+ pretoStart() : void
+ AddAllEntity(rsn : RegisterEntity) : void
+ positionLayout(rsn : RegisterEntity, row : int) : void
+ initRegisterEntities() : void
+ refreshMemoryTable() : void
+ initButtons() : void
+ getMemoryList() : void
+ convertPresuInsToMachineCode(ins : char) : char
+ start() : void
Other Instruction
+ getExecInfo() : void
MainMemory
+ getPrinterBuffer() : void
+ setPrinterBuffer() : void
+ setKeyboardBuffer() : void
+ MemoryManageUnit() : void
+ load() : void
+ readMemo() : void
Cache
- line : int
- tag : int
- offset : int
+ updateCache(add : char) : void
+ tryCache() : void
+ getCacheExecuteInfo() : void
+ updateByStore() : void
Trans
- binaryStringtoInt(bits:int,str:char) : int
+ transIntToBinaryString(i : int) : char
+ parseInsOfTYPE_A(String : char) : char
+ parseInsOfTYPE_B(String : char) : char
+ parseInsOfTYPE_ALU(String : char) : char
+ parseInsOfTYPE_IO(String : char) : char
<<interface>>
ReadAndWrite
+ write(address : int, value : int) : void
+ read(address : int) : int

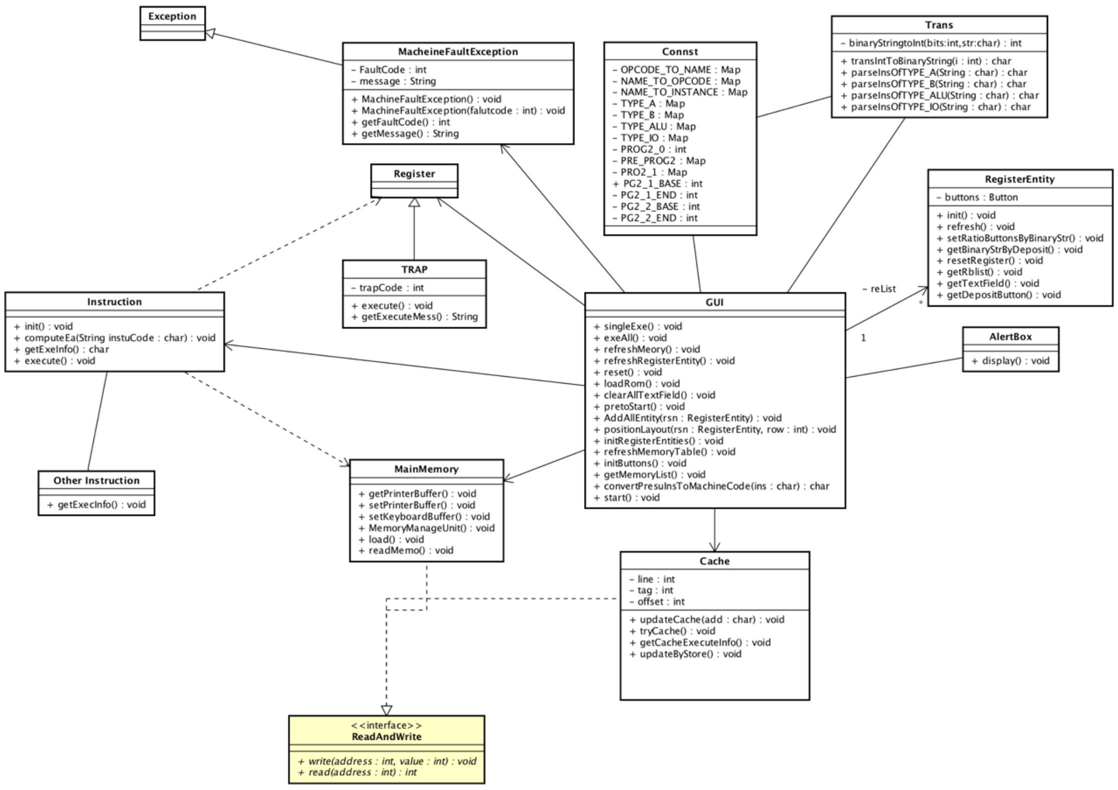
2.4 Update In GUI

In order to show the cache condition, the new graphic user interface add a new text area to show how cache work, it can show the conditions of “hit”, “miss”, and update. Besides, the new printer has a title to inform users that this can be used to show the execute information which bring a lot of conveniences for simulator test.

Additionally, we add the reset button to the gui, which can easily erase all information in the text field and reset all component in the gui to their original states. By doing this, the user need not to run the jar file for test another program.

3. Part 3

3.1 Update of class diagram


Exception
MacheineFaultException
- FaultCode : int
- message : String
+ MachineFaultException() : void
+ MachineFaultException(falutcode : int) : void
+ getFaultCode() : int
+ getMessage() : String
Connst
- OPCODE_TO_NAME : Map
- NAME_TO_OPCODE : Map
- NAME_TO_INSTANCE : Map
- TYPE_A : Map
- TYPE_B : Map
- TYPE_ALU : Map
- TYPE_IO : Map
- PROG2_0 : int
- PRE_PROG2 : Map
- PRO2_1 : Map
+ PG2_1_BASE : int
- PG2_1_END : int
- PG2_2_BASE : int
- PG2_2_END : int
Trans
- binaryStringtoInt(bits:int,str:char) : int
+ transIntToBinaryString(i : int) : char
+ parseInsOfTYPE_A(String : char) : char
+ parseInsOfTYPE_B(String : char) : char
+ parseInsOfTYPE_ALU(String : char) : char
+ parseInsOfTYPE_IO(String : char) : char
Register
RegisterEntity
- buttons : Button
+ init() : void
+ refresh() : void
+ setRatioButtonsByBinaryStr() : void
+ getBinaryStrByDeposit() : void
+ resetRegister() : void
+ getRblist() : void
+ getTextField() : void
+ getDepositButton() : void
TRAP
- trapCode : int
+ execute() : void
+ getExecuteMess() : String
Instruction
+ init() : void
+ computeEa(String instuCode : char) : void
+ getExeInfo() : char
+ execute() : void
- reList
*
1
GUI
+ singleExe() : void
+ exeAll() : void
+ refreshMeory() : void
+ refreshRegisterEntity() : void
+ reset() : void
+ loadRom() : void
+ clearAllTextField() : void
+ pretoStart() : void
+ AddAllEntity(rsn : RegisterEntity) : void
+ positionLayout(rsn : RegisterEntity, row : int) : void
+ initRegisterEntities() : void
+ refreshMemoryTable() : void
+ initButtons() : void
+ getMemoryList() : void
+ convertPresuInsToMachineCode(ins : char) : char
+ start() : void
AlertBox
+ display() : void
Other Instruction
+ getExecInfo() : void
MainMemory
+ getPrinterBuffer() : void
+ setPrinterBuffer() : void
+ setKeyboardBuffer() : void
+ MemoryManageUnit() : void
+ load() : void
+ readMemo() : void
Cache
- line : int
- tag : int
- offset : int
+ updateCache(add : char) : void
+ tryCache() : void
+ getCacheExecuteInfo() : void
+ updateByStore() : void
<<interface>>
ReadAndWrite
+ write(address : int, value : int) : void
+ read(address : int) : int