The George Washington University

Project 2: MIPS Object Oriented Design

By

Hangzhao Li
Joe Hallal
Mihir Mankad
Sogand Ghods

CSCI 6234
Professor Lancaster
April 29, 2019

TABLE OF CONTENTS

Statement of Problem

Issue:

The problem we are tasked with is developing an in-house version of a 32 bit MIPS simulator program, to be used in the Computer Education Corporations training centers. They will then expand upon this simulator by developing it into a design tool. We must design the complete OO design for this MIPS simulator.

Capabilities:

The prototype for the simulator must include add, subtract, load word, store word, branches with conditions, and absolute jumps. Our complete OO design should include Object Models, Communication Models, and State Transition Models. Another feature should be the components of user interface, and we must keep in mind the client may want to upgrade the functional organization later.

User Functions:

The user should be able to view the register set, code memory, data memory, and stalls. Our goal is to allow the user to navigate through the MIPS simulator efficiently, and our GUI should be easy to understand as well. The user should be able to see executed instructions, and the current execution phase for each instruction and any stalls that occur.

Assumptions

Assumptions for Architecture:

When designing the simulator, we first assumed that the core instruction set would be unchanging, and so our class diagrams would represent the total functionality of each machine process in the architecture. Another assumption we made was that Single Step Run operations would function exactly like normal execution, except with the addition of letting the user advance steps as necessary.

We interpreted the problem assignment, assuming it was asking for just basic functionality of MIPS, limiting some of the actions a user can take. We wanted the user to be able to run functions that can do the MIPS instructions in the set, but not allow for over complication of user code.

Our instruction set assumptions were that the machine had the ability to use each instruction in the set, but we didn't need to design the implementation of those instructions. We only had to describe their involvement/placement in the system.

Assumptions for Interface

We designed a simple interface that would allow the user to simply load a program in either single step or normal execution, view registers, and view certain data. We did not want to make the interface complicated as it seemed beyond the scope of what the problem statement was asking. We assumed the user would understand the basics of running a program in a MIPS simulator, and also the fact that debugging should be an easy task.

Processor State Diagram - 2

(**Team member 1**)

Case Diagrams : (Team Member 2)

User wants to load into simulator

1) User selects Load Program
2) System displays choose file menu
3) User selects file
4) System loads file and checks program input
5) System displays errors if needed
6) System updates memory display
7) System sits in idle

Exceptions:
4a) File cannot be opened. The file cannot be opened and an error message is displayed to the user. The use case ends.

Retrieve data from Simulator

1) User selects desired data
2) System loads file and checks file
3) System displays errors if needed
4) System displays file to user
5) System sits in idle

Exceptions:
3a) System retrieves corrupt data. User exits. The use case ends.

User runs program

1) User selects desired program file
2) User selects run
3) System loads file
4) System displays errors if needed
5) System runs entirety of program
6) The program terminates
7) System sits in idle

Exceptions:
3a) File cannot be loaded. System exits, use case ends.
5a) Program has errors and cannot run. The system pauses execution. Displays errors to user in the console. The use case ends.

User Clicks Load into Single Step

1) User selects desired program file
2) System loads file
3) User selects single step run
4) System displays errors if needed
5) System runs one instruction
6) User selects continue
7) Repeat 5) and 6) until termination or user clicks terminate
7) System sits in idle

Exceptions:
5a) If there is an error in program execution the system pauses execution. Displays errors to user in the console. The use case ends.
6a) The user selects exit and does not wish to continue execution. The use case ends.

User Views Registers

1) User selects desired program file
2) System loads file
3) User clicks view registers
4) System displays register contents
5) System sits in idle

Exceptions:
4a) The register is empty. Tells user register is empty in console. The use case ends.

User Views Memory Contents

1) User selects desired program file
2) System loads file
3) User clicks view mem
4) System displays stack with data
5) System sits in idle

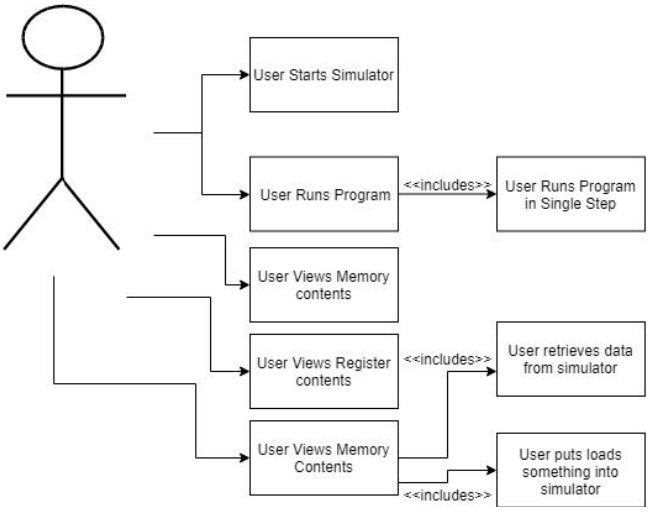
Exceptions:
4a) There is a corruption in memory. The system displays corrupt data followed by an error message. The use case ends.

User Starts Simulator

1) User launches simulator
2) System launches into boot
3) System interface displays to user
4) System sits in idle

User Exits Simulator

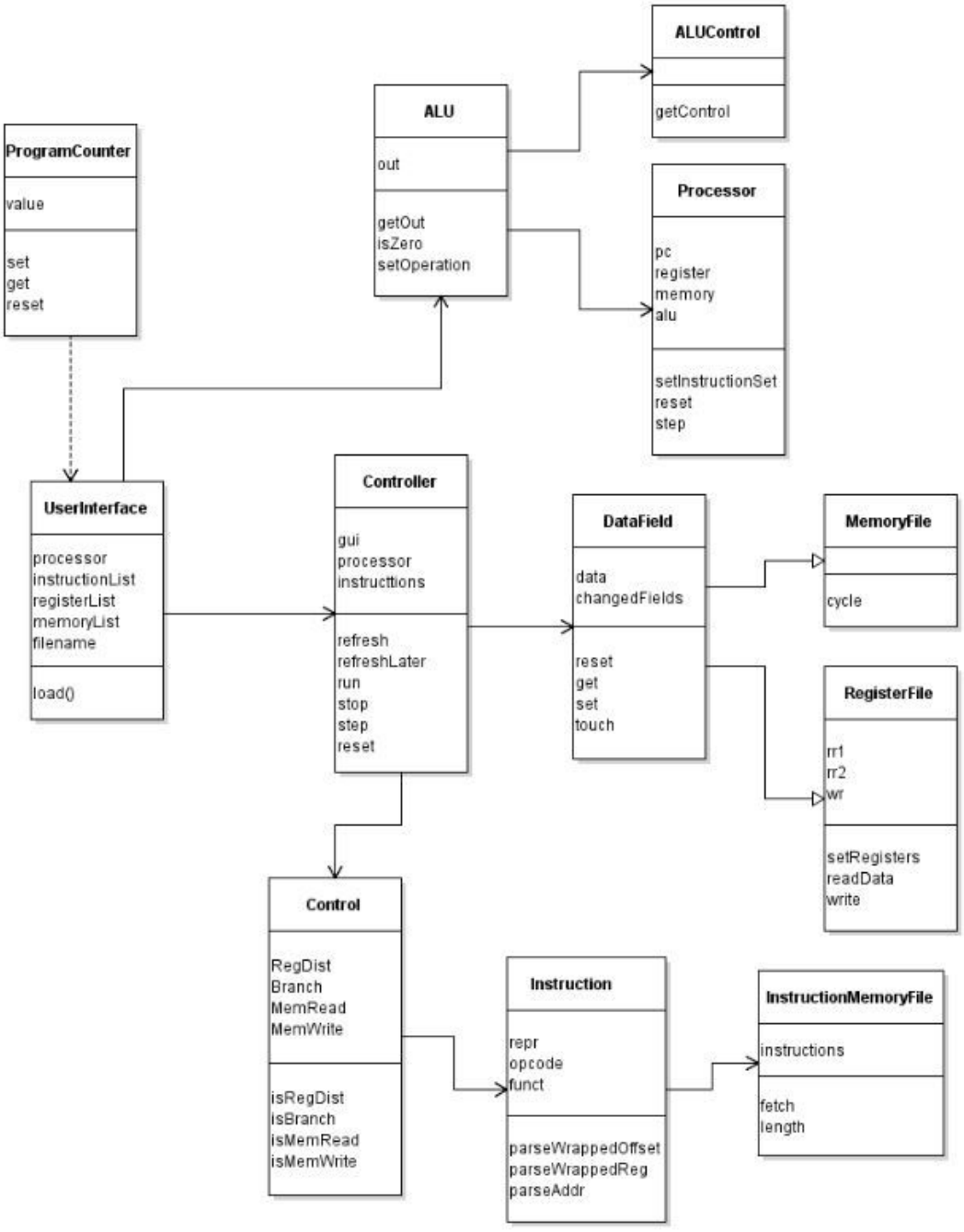
1) User selects exit simulator
2) System prompts user to confirm quit
3) User selects confirm
4) Simulator terminates



(**Team member 2**)

Processor State Diagram - 2

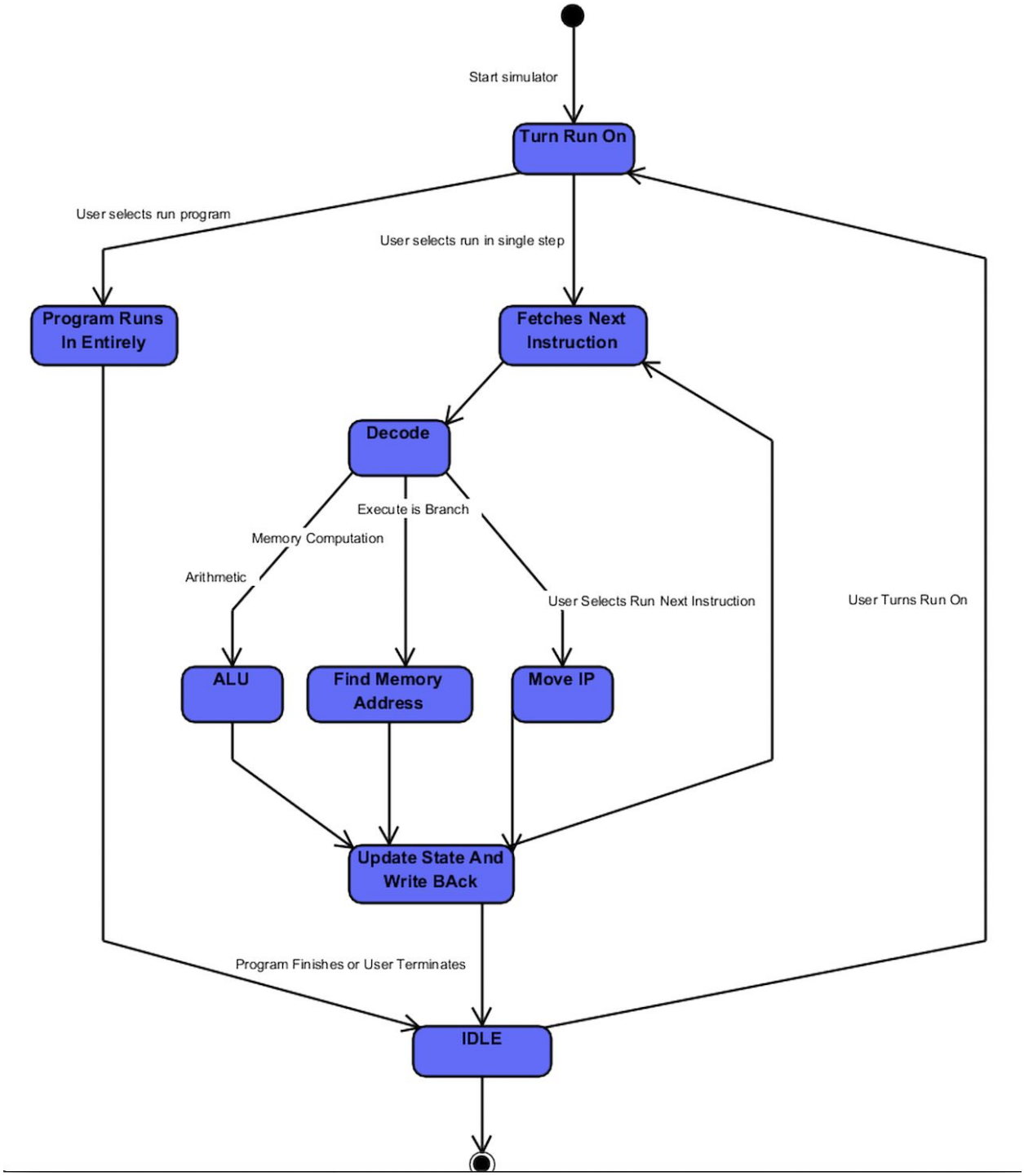
Class Diagram



Processor State Diagram - 2

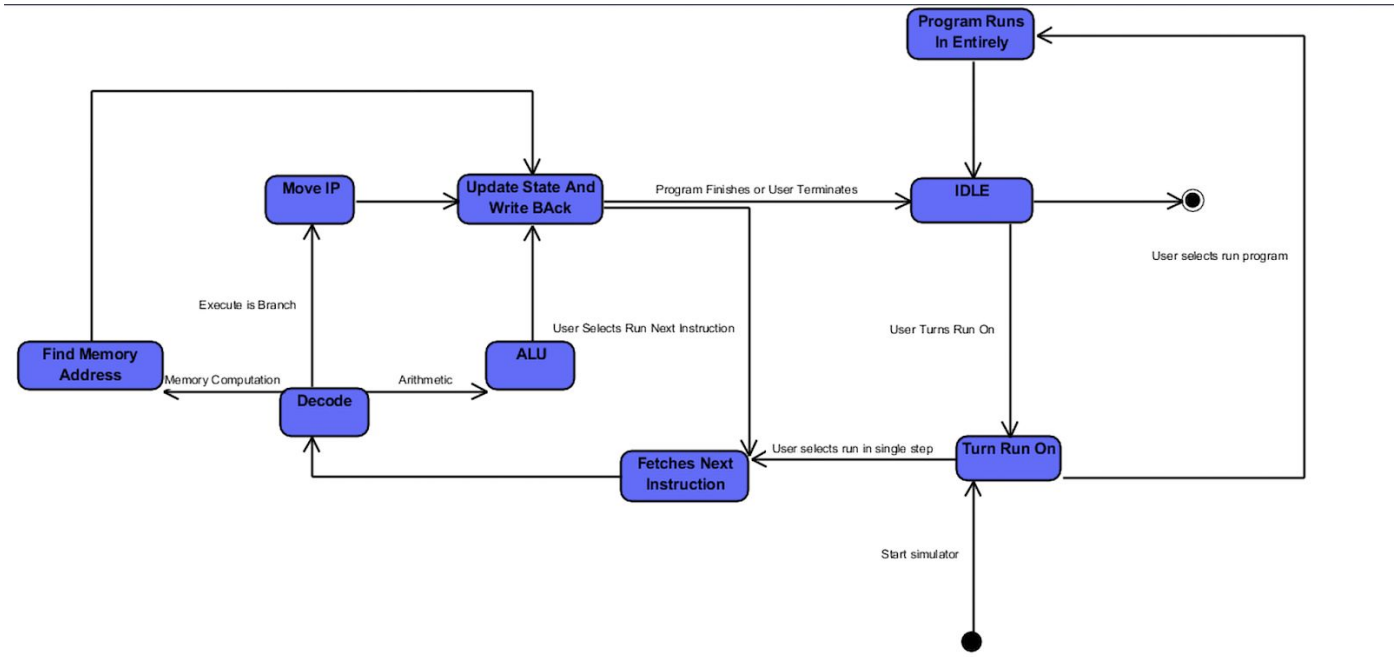
(Team member 3)

State Diagram



Processor State Diagram - 2

1-1



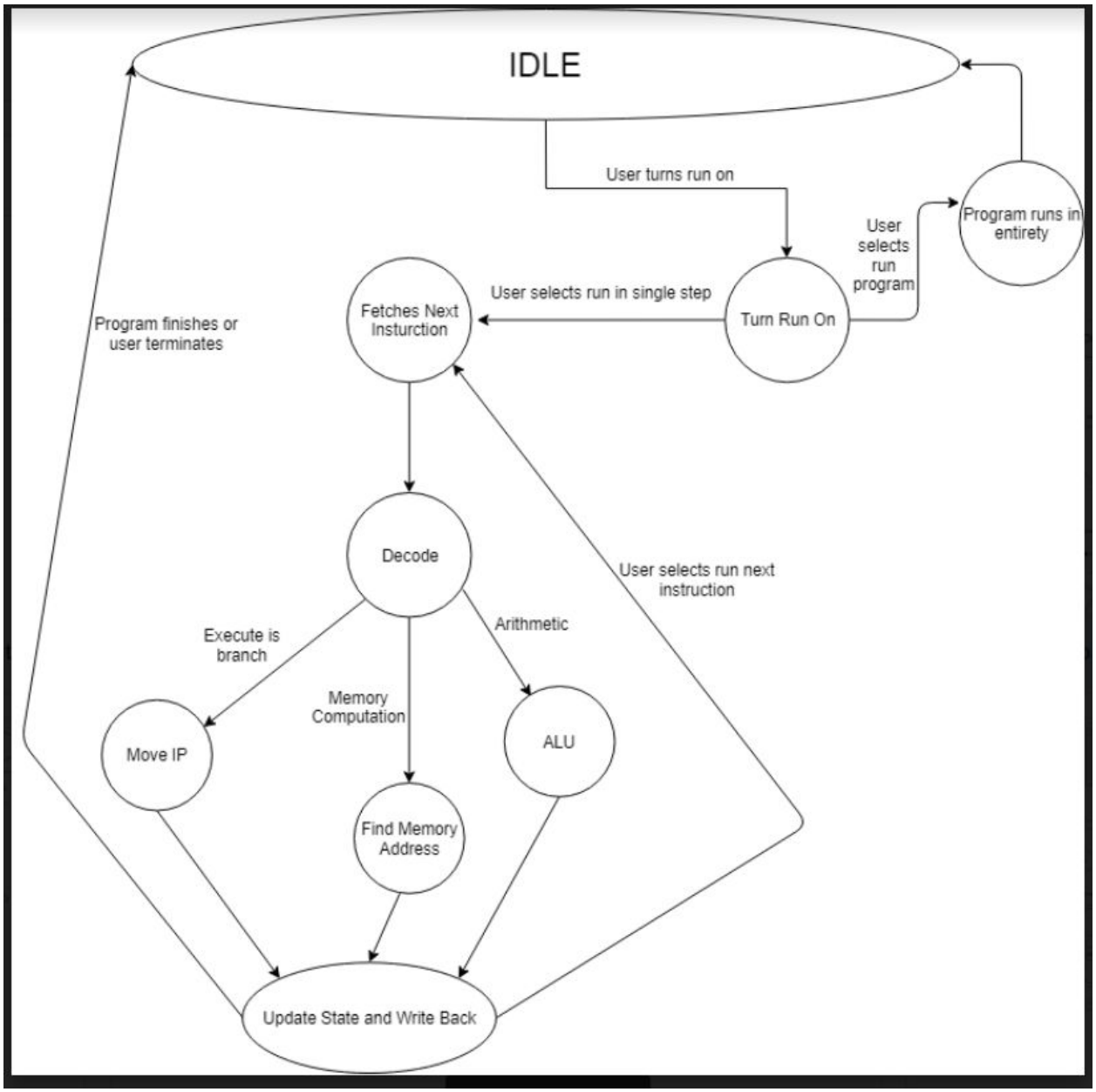
Full state diagram

Processor State Diagram - 2

(**Team member 4)**

(Diagram P1) (Team Member 2)

Processor State Diagram - 2



(Team Member 2)

Processor State Diagram - 2

Diagram P2 (User Facing Flow)