Component Based MMIX Simulator using Multiple Programming Paradigms

Stephen Edmans

Contents

List of Figures

1 Introduction

As software systems get larger and more complex there is a need to handle this complexity. There is a prevailing design paradigm, which addresses these issues, that is to break these systems up into smaller components. This is a sentiment mentioned by Turner [3] He calls these components "collections of modules", These components will interact with each other to make the complete system.

When you have control over the development of more than one of these components it is traditional to use a single programming paradigm for your components. There is, however, no reason that you cannot use different languages and paradigms for these components. The goal of this project is to create a relatively complex system that is made up of multiple components where each component uses the most appropriate programming paradigm for each component.

The system that I plan to create in this project is inspired by Jeliot [4], which is a tool that is used as an aid in the teaching of Java. The Jeliot system allows a user to give it a piece of Java source code and it will show the user what the underlying virtual machine is doing when it runs the code.

In his seminal work The Art of Computer Programming [1] Dr. Donald Knuth designed an artificial machine language that he called MIX. In a later volume of his work [2] Dr. Knuth updated this machine architecture, which he calls MMIX. This project will create a system that take MMIX assembly code and shows the user, graphically, what the simulated machine is doing.

2 Existing Implementations

There is already one existing main implementation created by Dr. Knuth and his team at Stanford University. This implementation runs as a few command line utilities. These utilities include an assembler which converts a text file containing the relevant assembly language source code into a proprietary binary format. There is a utility which reads files in this proprietary binary format and displays the content back to the console. There is also the main utility which will execute the application stored in the binary files using an MMIX virtual machine. It is possible to get the virtual machine to output some tracing information but it does not have a graphical user interface and it does not allow you to step through the processing of the application.

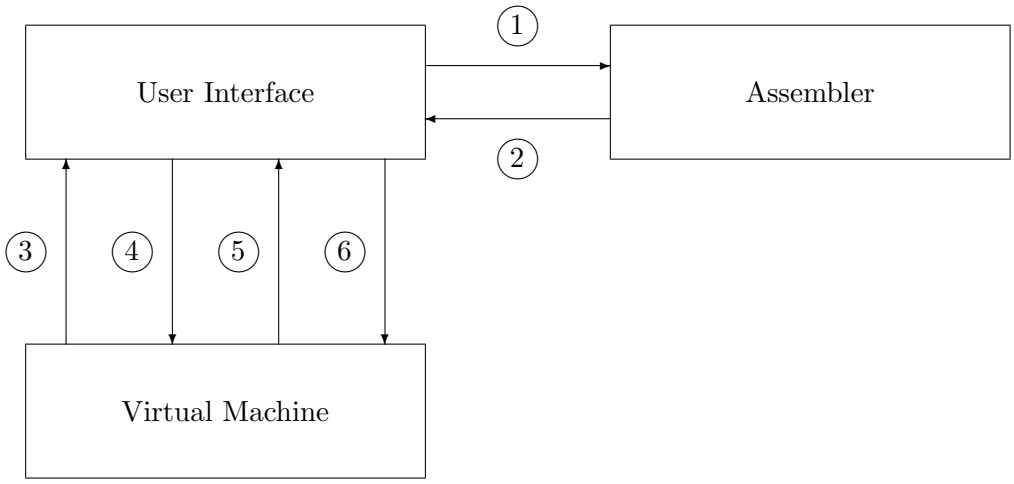
These utilities are all written in CWEB

3 Outline Design

The MMIX simulator application will consists of three separate components. The task of converting the MMIX assembly language source text into the corresponding binary representation will be performed by a component I am calling the Assembler. The task of actually simulating an MMIX computer will be performed by a Virtual Machine component. The final component will be responsible for representing the current state

of the MMIX computer to the user, along with orchestrating all of the interactions with the other components. I am calling this component the User Interface. The interactions between the components can be illustrated by figure 1.

One question that needs to be answered is what is the target platform for this application. The application will initially be written for Mac OS X but if there is time available at the end of the development process I would like to make sure this application will run on other platforms.



1. Source Text
2. Binary Representation
3. Binary Representation
4. Current State
5. Process Next Step
6. Change of State

Figure 1: Component Interactions

3.1 Assembler

The purpose of the assembler component is that it will take in MMIX assembly language source code contained in a source file and convert it into a binary representation of the application. The first thing that I need to decide upon, for each component, is what is the most appropriate programming paradigm. The production of an assembler is a fairly well defined process. There should be little, or preferably, no ambiguity in the translations. These requirements make me think that this would be an ideal candidate to be written in a functional language, which to quote Turner [3]

> Functional programming is so called because its fundamental operation is the application of functions to arguments. A main program itself is written as a function that receives the program's input as its argument and delivers the program's output as its result.

There are many functional languages available so determining which one to use requires some consideration. One way to classify functional languages is to determine how easily they allow users to create side effects. A procedure performs a "side effect" when a procedure does not just return a value it also amends the underlying state of the system. Functional languages where you have to explicitly declare that code can perform side effects is called "pure". I plan on using a "pure" functional programming language for the assembler.

3.2 Graphical User Interface

The main way that users will interact with the simulator is through a graphical user interface (GUI). The GUI will be responsible for all of the interactions between the other components. The GUI will initially be written to run as a stand alone application. If there is time at the end of the project I will investigate how to push the GUI up onto the internet or even a mobile version.

The only consideration that needs to be taken into account when deciding which programming paradigm and language to use when creating this component is that a GUI, almost by definition, has to handle numerous "side effects". This would make me lean towards an object oriented programming language, but there is a newer family of programming languages that allow the developer to use multiple programming paradigms. I plan on using one of these multi-paradigm programming language for the GUI.

3.2.1 Initial Design

The GUI will be broken up into four separate main sections as illustrated in figure 2. Each of these sections will contain details about one distinct arrear of the MMIX simulator.

3.2.2 Memory

The memory section of the GUI will contain a representation of the simulators current memory. The memory representation will show both an hexadecimal representation for

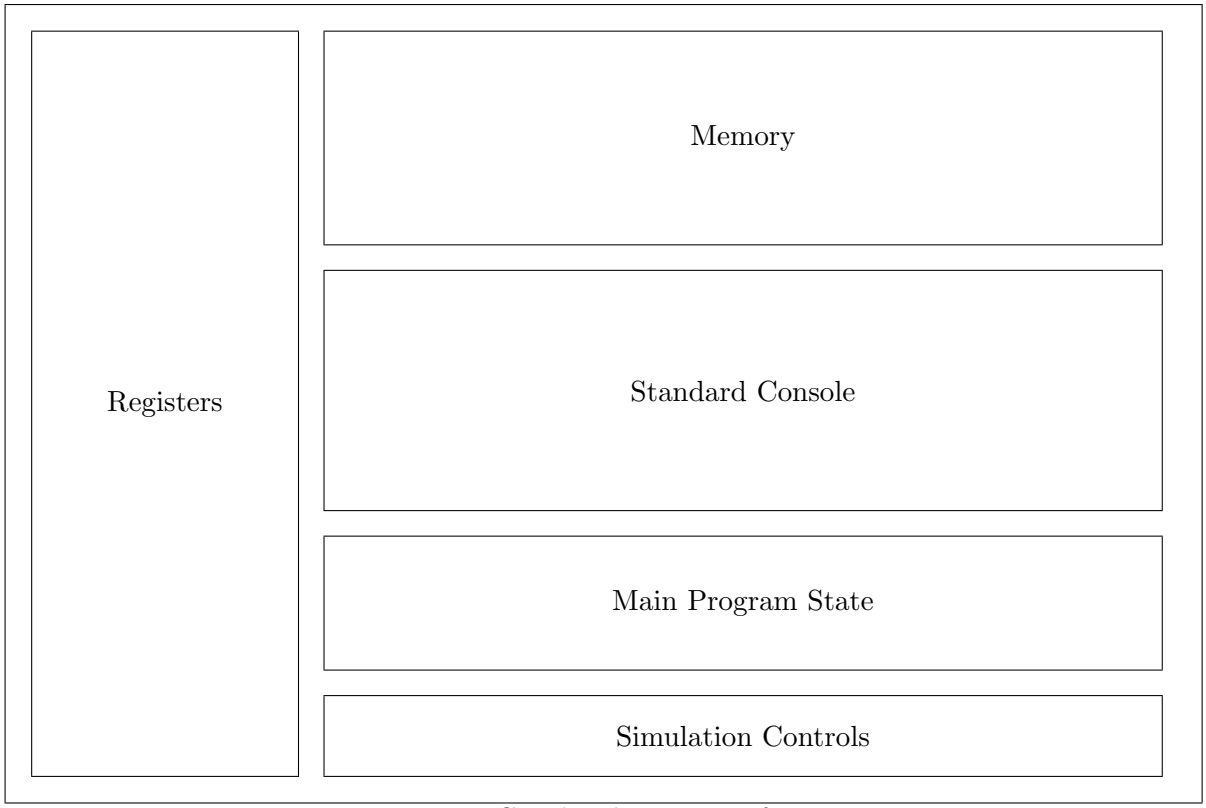


Figure 2: Graphical User Interface

the values in memory but also an ASCII representation of these values. An example of what I am planning to include can be found in figure 3. I am envisaging that when the contents of a specific memory address changes then that location will be highlighted, probably with a change of colour. After a period of time this highlighting will be removed. This period of time will probably be when the user executes the statement in the program.

00000000 :	$8fff$	0100	0000	0701	$f4ff$	0003	0000	0702	.....
00000010 :	0000	0000	$2c20$	$776f$	$726c$	$640a$	00		$...., world..$

Figure 3: Memory Representation

3.2.3 Registers

The MMIX architecture contains two types of registers, general purpose registers and special registers. There are 256 general purpose registers that the developer is free to use as they see fit. There are 32 special registers which either record the internal state of the MMIX machine or how the MMIX machine is configured. The registers section

of the GUI will display a list of all of the available registers in the simulated MMIX machine. It will show not only the names of the registers but there current values. I am envisaging that when the contents of a register changes then, like the memory section, it will be highlighted.

3.2.4 Main Program State

When every statement in the program is executed then the internal state of the MMIX machine will change. I want the user to clearly see what these changes are. One way I am intending to show these changes is with the highlighting of memory addresses and registers that I have mentioned above. Another way I am intending to show these changes is with this section. This section will list all of the registers that have changed along with possible a sample of the changed memory addresses.

3.2.5 Standard Console

There are a number of embedded computers that do not interact directly with a user however it is quite rare for a general purpose computer to have no interactions. The MMIX simulator has got standard input and output channels. These will need to be accessed in the simulator. They will be accessed in the simulator through the standard console section.

3.2.6 Simulation Controls

The users will need some way to interact with the MMIX machine. The simulation controls section will allow the user to control how the GUI communicates with the virtual machine. This will contain all of the interactions detailed in the virtual machine section (section 3.3).

3.3 Virtual Machine

The Virtual Machine component is the heart of the simulator. It is this component that actually simulator the MMIX machine. It will contain a representation of the MMIX machines memory. It will also contain all of the registers that the MMIX machine uses. I am envisaging that the application programming interface, which to quote Wikipedia [5]

> An application programming interface (API) is a protocol intended to be used as an interface by software components to communicate with each other.

for the VM will be very small.

The way that the virtual machine will change it's state is when it receives a command from the outside world. It will sit there waiting for the next instruction. The virtual machine will respond to these commands by performing the relevant actions and returning details of any changes to the virtual machine. This leads me to want to use a functional language for this component and it also makes me wants to use a language that has got strong message oriented features.

3.3.1 Application Programming Interface

I am envisaging that the application programming interface for the virtual machine will only contain the following commands:

Reset Simulator This command will close down the existing virtual machine and await for a new program to be specified.

Load Program The GUI will pass the binary representation of a program to the virtual machine which will load this into the memory representation and it will reset the registers.

Reset Program This command will reset all of the registers in the virtual machine.

Process Next Statement This command will execute the current statement and move the state of the virtual machine on to the next statement.

4 Development Methodology

5 Development Plan

6 Summary

References

[1] Knuth, D.E., Art of Computer Programming

[2] Second Volume

[3] Turner, D., 1990. Research Topics in Functional Programming, Addison-Wesley. Available through <`http://www.cs.utexas.edu/~shmat/courses/cs345/whyfp.pdf`>[Accessed 2 April 2013]

[4] University of Joensuu, 2007. Jeliot 3. [online] Available at: <`http://cs.joensuu.fi/jeliot/`>[Accessed 5 April 2013]

[5] Wikipedia. Application Programming Interface [online] Available at: <`http://en.wikipedia.org/wiki/Application_programming_interface`>[Accessed 7 April 2013]