CMP2089M, Instruction Set Emulation - Assignment 2

Group 29 - Alex Howe, Hayden Moir, Lokesh Bhatti, Matthew Micklewright, Samuel Rehill and David Churchill
Project Supervisor: Dr Charles Fox

University of Lincoln, School of Computer Science

2019
April

Abstract

This project has been aiming to emulate the ARM7TDMI processor (as well as its instruction set) in software as laid out in our project proposal. The ARM7TDMI processor which was designed by ARM holdings and licensed for manufacture by Texas Instruments (among another 164 licensees) in 1993, was most famously used in the Nokia 6110 phone [1]. The ARM7TDMI is a RISC (Reduced Instruction Set Computer) architecture and features a 32-bit Data bus, 32-bit Address bus and 32-bit ALU (Arithmetic Logic Unit). As well as 32-bit ARM instructions (common to all ARM processors) the ARM7TDMI also has the capability to run the Thumb instruction set. These are 16-bit instructions that upon execution are decompressed into full 32-bit ARM instructions and, as such, can be used in situations where code density is preferable to execution speed. The ARM7TDMI has 7 operating modes, 7 interrupt/ exception types and a 3-stage pipeline as well as being a Von Neuman Architecture [2].

Contents

1 Introduction

1.1 Project Overview

This project initially set out with the aim to enhance our own understanding of RISC architectures (and in particular the ARM7TDMI). However, as the team proceeded with creating the emulator it was soon realised that not only could this be an effective means to test our understanding of the architecture, it could also be a tool to help others visualise the workings of the processor.

The aims of this project therefore became two fold: Firstly, to enhance all group member's understanding of the design and functionality of RISC architectures and secondly, to create an emulator that could help to demonstrate a RISC architecture to others.
Subsequently, there were a number of objectives to fulfil to ensure these aims were achieved, the prime one to create a working emulator in software of a RISC architecture (and in particular the ARM7TDMI). The first step in this was research into RISC architectures in general as well as the exact specifications of the processor.
A full review of academic literature was performed in our project plan, however our key sources were:

1. Stallings, W. (2016) *Computer Organization and Architecture: Designing for Performance*, 10th Edition. Boston: Pearson.
2. ARM ltd. (2004) *ARM7TDMI: Technical Reference Manual*, Revision: r4p1.
3. ARM ltd. (2005) *ARM Architecture Reference Manual.*

The first gives a broad overview of the topic of Computer Architecture and was a vital starting point. The second and third give technical details on ARM instructions and the ARM7TDMI's own specific architecture.

Once research had been done on the requirements to emulate the processor the work was divided into atomic packages that could be completed by a subset of the group.
Table 1 shows the work packages, the importance attributed to them in the project plan and the package's completion date. Of particular note is the 'Reverse Debugger' which was added later, when the project's aims were broadened to include it acting as a teaching tool.

In terms of application, this could easily be used in a classroom environment scenario to demonstrate the effects and usage of assembly code and lower-level languages to undergraduate students or people studying Computer Science at Sixth Form or GCSE. With more work it could possibly be utilised to show the workings of CPU bugs such as the Spectre and Meltdown vulnerabilities.

Note that, to assist in the marking of this document, extracts from the criterion reference grid have been included as margin notes beside the section(s) that are evidence of fulfilling that criteria.

Criterion 2: Document the production of the artefact in written or video form.

Criterion 2: Changes made to the aim and objectives have been fully explained and justified

Table 1: Work Packages and information about their completion dates

Work Package	Importance	Goal Delivery Date	Actual Delivery Date
Registers	High	18/02/2019	18/02/2019
Control Unit (with LDR, STR & ADD)	High	08/03/2019	08/03/2019
ALU	High	08/03/2019	08/03/2019
Memory	High	08/03/2019	08/03/2019
Branching	Medium	22/03/2019	22/03/2019
Other ALU Instructions (MUL, SUB, ORR etc.)	Medium	22/03/2019	08/03/2019
Addressing Modes	Medium	22/03/2019	29/03/2019
Interrupts	Low	05/04/2019	N/A
Operating Modes	Low	05/04/2019	N/A
Thumb Instructions	Low	05/04/2019	N/A
Pipeline	Low	05/04/2019	N/A
Reverse Debugger	N/A	N/A	05/04/2019

1.2 Group Process

When deciding upon the process for this group project the two most popular (and opposing) frameworks, Agile and Waterfall, were considered. In reality these two have a variety of extensions and sub-cultures but the general principles they encompass stand at either end of a spectrum. For the purposes of this project however, they were considered as discrete and atomic frameworks.

Criterion 2: Theoretical frameworks presented in the lecture programme are critically analysed and related to actual group experience. Reference is made to other theoretical frameworks and their specific relevance to a group project is discussed.

Waterfall is a framework where "the overall process generally cascades down" [4]. It works from the principal that once you have the requirements of a system, these can be broken down into specifications which outline fully how the "team will be able to design, build, and/ or purchase subsystems and components" [4] Waterfall has the benefit of leaving a clear trail of documentation from the initial requirements, to a full technical specification. It does however have some drawbacks, primarily it can suffer 'scope creep' i.e. initial requirements are established but, as the design progresses, the aims and requirements shift. This is a great concern due to waterfall's rigidity and the core idea that 'work can only cascade down'. If changes are made, it requires that the process must be restarted from the point of alteration.

Agile's main principles are "Individuals and interactions over processes and tools, Working software over comprehensive documentation and Responding to change over following a plan" [3]. It aims to fix this issue of "scope creep" by fully embracing it; Agile assumes that requirements will change and so avoids ever having a large overarching plan.

"The project is divided into a sequence of small, iterative efforts, each conducted by a team devoted to meeting a limited set of requirements and releasing a partial result or solution" [4]. This allows for additional requirements to be added at any stage of the process. For example, in this project it was realised that, to enhance the emulator's capabilities as a teaching tool, a reverse debugger would be highly beneficial. In a waterfall framework this would either have meant returning to the design stage, abandoning most of the implementation done or simply continuing without this feature.

As stated by Agile Alliance above, keeping a working prototype available at all times is a vital feature of Agile and was something the team found useful whenever a demonstration of progress to the supervisor was needed.
There are however drawbacks to the Agile framework, because there is never confidence in the current state of the project, there is no incentive to ever create documentation. This can lead to a situation at the end of the process with little, or no reliable documentation.
It can also be said that because there are no concrete requirements in a project that it is hard to say when it is complete, it is possible that a system might forever receive new requirements and functionality.

Of course a variety of other frameworks, many based upon the core ideas of Agile and Waterfall. For example, Scrum is a framework which builds off of Agile, it works on the basis of having 'Product Owner' who coordinates small, "cross-functional and self-organising teams" [7]. The framework is divided into short 'sprints' with a small sub-aim at the end; the sprints last for around two weeks and whilst they are on the team has a daily stand-up meeting tp "review what they've done and what they will work on for the rest of the day" [7].

2 Solution design

2.1 Justification of Tools

Criterion 2: The choice of tools used in the production of the artefact is completely appropriate and has been comprehensively justified.

The first tool is a shared OneDrive Site, in conjunction with Microsoft Word which was used to write the project plan. Having the site allowed us to easily share any documentation (of almost any file type) found during the research phase of the project, once the project moved onto compiling a formal plan, collaborative use of Word Online allowed for the editing of the same document in real time from different locations. Another benefit is the automatic cloud backups which reduce the risk of data loss (a constant concern for any digital project). There are however downsides to Word Online; its design as a word processor it is unable to typeset as well as a program designed with typesetting in mind.

Criterion 2: Clear evidence of the way in which the tools selected were used to support group processes has been provided.

Due to the decision to run this project as an iterative process in line with Agile development, it would be beneficial to use a version control system so that experimentation could be done on new branches whilst maintaining a working master branch for demonstrations. GitHub was used due to its centralised cloud storage, version control and functionality for team working. Another advantage of using GitHub has also been able to run a vast range of different programming languages making it very useful when merging together different types of code, an example of this is when C++ was used to create the instructions sets and emulated parts of CPU, then Javascript used to show a visual front end to show of the process of the emulation. There is also a revision control system which

keeps a record of each persons work contributed to the project which is very helpful to identify people who are not actively involved and prevent worries of group members taking credit for other people's work.

To be an effective team, it is important to maintain good and clear communication between one another. To do this, a text message service that is free to use was needed. There were a variety of options available but in the end, we decided to choose Discord. The reason we chose Discord is that it is a supporter of open source software development and is itself, open source [10]. It allows for group conversation and uploading files while still being a fast and responsive text messaging service. Discord can also be used on various devices including within web browsers, and on phones making it more accessible. We also used Discord to log our group meetings because this provide more transparency to what is happening in each of the group meeting and what our aims were for the next meeting. Discord is a good communication device, but it does have some limitations such as that it cannot send large text files, for this reason we also used email to send files and letters. Furthermore, email can also be used for backup just in case we lose communication through Discord.

When deciding upon the programming language(s) used in this project, it was decided that it would be beneficial to use a language with which the team were all familiar. This caused to choose C++ as our main language, the reason for this choice is that it is an object-oriented focused language which means the way it would be organised and structured will make it more intuitive. C++ programs can be broken down into separate source (and header) files meaning the programming can be split into parts, making it better for Agile development. An example of this was creating the four classes: registers, control unit, ALU and memory (code available in appendix A.7 & A.8, A.1 & A.2, A.3& A.4 and A.5 & A.6 respectively). Another advantage is that C++ also use inheritance and polymorphism which allows reusing code to increase efficiency and save time. C++ will also provide more experience in the computer science industry, according to Stack Overflow's 2018 survey of professional developers 24.6% used C++ [6].

During the development, it was decided that a more visual representation of the emulation was needed, unfortunately, C++ has limitations in terms of visual display which is why Javascript was used. Another reason why to use Javascript was the high amount of interactivity, making the emulation more user friendly which would help the student get more involved in the program. The third reason why Javascript was a good choice is it has good interoperability making it workable on a variety of operating systems.

Another challenge was finding a suitable IDE (Integrated Development Environment) that will increase the productivity of coding and help spot programming errors. Our preferred tool was Visual Studio, the reason is Visual Studio's graphical user interface is designed for complex codes like C++ and JavaScript

making it much easier to read and modify. Another advantage of using Visual Studio is that it has an inbuilt debugger, making it much easier to spot programming errors and find ways to improve efficiency.

Despite using Word Online to write our first report, it was decided to switch to a tool that was better at typesetting. Due to it being a popular, open source tool, LaTeX was an obvious choice. Latex is also commonly used in technical papers, reports and books which mean that it is useful to learn as it can be applied to many common industry practices. The main drawback of LaTeX, is that nobody in the team had any prior experience using it, so it could be argued that a small amount of time was wasted getting up to speed on how to use the software.

2.2 Program Structure

Criterion 2 & 3: Document the production of the artefact in written or video form

To better increase the chances of conveying an accurate understanding of the workings of the processor and its sub-components, it was decided that designing the code in an object oriented style was key. The reason being that when explaining a processor, it is often by describing atomic components like the ALU and Control Unit and the way they interact and perform tasks independent of each other. This translates well to an Object Oriented style where there can be separate objects interacting with each other.

The general structure of the objects as indicated in Figure 1 is that the control unit is the top level object, it in turn then possesses subcomponents: memory, an ALU and an array of Registers (36 registers and one program counter). As in the physical processor, the control unit handles the fetching of instructions from memory, decodes them and executes them within the ALU.

Fetching is done in *MainTest.cpp*, decoding and then executing the instructions is then done inside of the Decode subroutine in the *ControlUnit.cpp*, the full code for this process can be found in Appendix A.2.

The Decode subroutine collects the instructions and then runs them through the process of testing for a valid condition and/or instruction, if the condition is valid it is checked. Instructions that make it this far are then put through a large 'if' 'else' condition to find the corresponding function, the instructions functionality is then performed (e.g. set memory location if "STR"). Arithmetic and logic instructions are passed to the ALU object which handles instructions in a similar way.

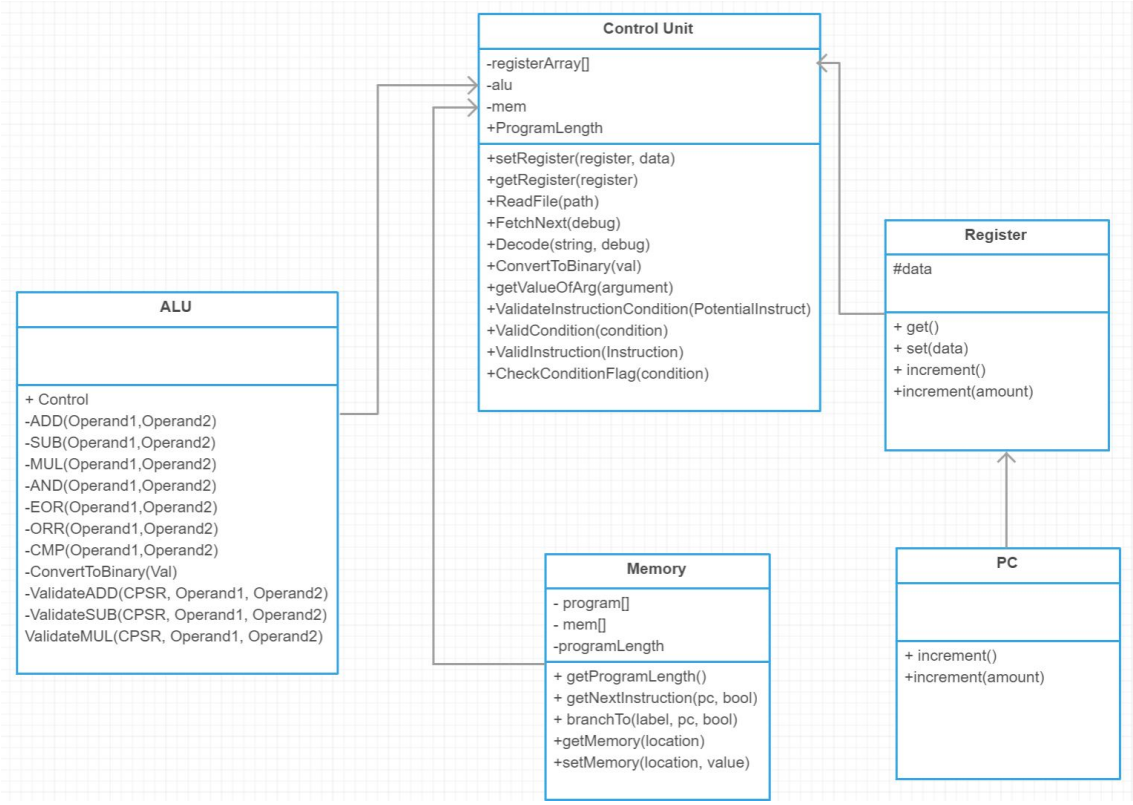
When the ALU receives an instruction (see Appendix A.4) in the *Control* subroutine it compares the instruction to a list of valid instructions in an 'if', 'else' statement. The corresponding function is then performed for that instruction.

Because all ARM instructions are conditional, a check has to be done in the decode phase to not only root out incorrect instructions and/or conditions but

also to perform the conditional check if one is required. The subroutine *ValidateInstructionCondition()* takes the potential instruction to be run and checks whether it is a valid instruction and/or condition. It does this by using the two functions *ValidCondition* and *ValidInstruction*, the former takes the last two characters of the string and checks them against a list of valid conditions.
ValidInstruction takes all but the last two letters of the string and checks against a list of valid instructions to ensure it is correct. By splitting off the string into these two functions and returning a Boolean value for each, it is ascertained whether there is an executable instruction on its own or an executable instruction with a condition. The source code for this processes is available in Appendix A.2.

Another area of particular interest in the program is the handling of memory; in the physical processor there is a 32-bit address bus allowing for an addressable memory space of 4,294,967,296 bits. However, to reduce complexity it was decided that instead of a fixed memory, a variable sized vector to simulate memory could be used. LDR pops from memory vector and STR pushes.
For the most part, register values are stored as an Integer data type, however, when manipulation of a single bit is required, such as when the ALU performs a comparison (CMP) it must alter the zero flag of the CPSRs (Current Program Status Registers) the Integer value must first be converted to a 32-bit binary number. The code for this is available in Appendix A.3 in the *ConverToBinary* subroutine. Once the value is converted the n th bit can then set to True of False.

Figure 1: UML Diagram Explaining Class Structure



2.3 Emulation Vs Simulation

In a project such as this, a key issue is the debate between emulation and simulation. Is it better to faithfully reproduce a system at the expense of simplicity and maintainability?
Generally, the view that faithful emulation is best, unless it will cause part of the system to be needlessly intricate and subsequently hamper a student's understanding, has been taken. For example, there are 37 registers capable of storing a signed 32-bit binary number, just as in the physical processor, however they are treated as Integer data type unless it is absolutely necessary to address individual bits (for example in the program status registers).This is because for the most part, the register values are more understandable as integers to a human programmer.
Another example is the use of a vector to represent memory, it is not of fixed size as the physical memory would be but it reduces file size and complexity when addressing it and thus is a reasonable cause to simulate this section rather

than directly emulate it.

3 Evaluation

3.1 Functional Demonstration

Criterion 1: Produce and demonstrate an artefact which satisfies project objectives and achieves the project aim.

A collection of images demonstrating the working emulator are available in appendix C.
Figure 2 shows the program running, on the left hand side is a visual representation of memory. In the top right are the registers (R1-37) and in particular the Current Program Status Register's (CPSR) value in binary is highlighted in blue to allow the programmer to check the flag bits.
In the bottom right is the program being run, the blue highlight indicates the instruction that has just been executed. Next to this are the controls so that a programmer can step through the program line by line (these are shown better in Figure 9).
In Figure 2, the program being run generates the Fibonacci sequence (a clearer version is available in Figure 7). On line 5 of this program there is a CMP (Compare) condition; in an ARM7TDMI the Zero flag of the CPSR is set to true when a CMP condition is met. Figure 10 shows the updated CPSR from that in Figure 2, the 30th bit is now set to one.

3.2 Reflection on Group Process

Criterion 3: Critically reflect on group processes and analyse them with reference to aspects such as Belbin and Tuckman and other Group Working processes.

When an individual or sub-group found itself struggling with a task they delegated it, of course "you should not delegate those things that you are good at doing"[5], and the team always encouraged each other to select tasks they felt capable of accomplishing individually or as part of a sub-group.

There were times where our deliverables were behind the planned date of completion. For example in a meeting, the minutes of which are available in appendix B.11, there was a discussion about issues a sub-group were having completing the addition of addressing modes to the emulator. In a waterfall framework this could have been very disruptive to the pre-planned schedule. However, due to Agile's flexibility extra team members were able to join that sub-group to work on the problem, the deadline was also re-adjusted. This is a prime example of how Agile can work around unforeseen difficulties.

As can be noted in table 1, some of the work packages were not delivered due to time constraints. Functionality such as interrupts, operating modes and pipelining were deemed as less important as the project progressed. In particular, it was decided that adding a reverse debugger was of much higher importance in achieving our aim of a teaching tool for RISC architectures than these wok packages.

Criterion 3: A good reflection on group processes and how their improvement might have led to an improvement in project outcome.

In conclusion, this project has been executed well. We utilised an agile development model, successfully used GitHub as a repository for our code and as a system for version control and achieved a suitable balance between emulation and simulation. With more time and resources, this project could have created a more faithful and accurate emulation of the ARM7TDMI, however the current implementation sufficiently meets the project aims.

Were this project to be repeated, an additional goal would be to aim to have more documentation on the structure of the program as, due to using Agile there was never particularly an incentive to document anything. Another area of improvement could be communication between group members; there were times when group members did attend consecutive meetings leading to a lack of cohesion on the project's direction.

3.3 Individual Contributions

Table 2: All group members along with their agreed contribution

Name	**Student ID**	**Contribution**	**Signature**
Lokesh Bhatti	16609087	16	
David Churchill	17642848	20	
Alex Howe	15618835	16	
Matthew Micklewright	16626154	16	
Hayden Moir	16608564	16	
Samuel Rehill	17638743	16	

References

[1] Arm Commuity. *A Historic look at Arm holdings from 1990-1997*. [online] Available at: https://community.arm.com/processors/b/blog/posts/a-brief-history-of-arm-part-1 [Accessed 20 Feb. 2019].

[2] ARM ltd. (2004) *ARM7TDMI: Technical Reference Manual*. Revision: r4p1.

[3] Agile Alliance. (2019). *Agile Manifesto for Software Development*. [online] Available at: https://www.agilealliance.org/agile101/the-agile-manifesto/ [Accessed 6 Apr. 2019].

[4] Nicholas, J.M and Steyn, H (2017) *Project Management for Engineering, Business and technology*. Fith Edition. New York, USA: Routledge Ltd.

[5] Belbin, R. (2010) *Team Roles at Work*. second edition. Oxford, UK: Elsevier Ltd.

[6] Stack Overflow. (2019). *Stack Overflow Developer Survey 2018*. [online] Available at: https://insights.stackoverflow.com/survey/2018 [Accessed 9 Apr. 2019].

[7] Paymo. (2019). *Project Management Methods, Methodologies, and Frameworks*. [online] Available at: https://www.paymoapp.com/academy/project-management-methodologies/sixsigma [Accessed 9 Apr. 2019].

[8] Stallings, W. (2016) *Computer Organization and Architecture: Designing for Performance*. 10th Edition. Boston: Pearson.

[9] ARM ltd. (2005) *ARM Architecture Reference Manual*.

[10] Discord. *Discord Loves Open Source*. [online] Available at: https://discordapp.com/open-source [Accessed 11 Apr. 2019]

A Source Code

A.1 ControlUnit.h

```
#pragma once
#include <vector>
#include <string>
#include "Register.h"
#include "Memory.h"
#include "ALU.h"
using namespace std;

class ControlUnit {
private:
vector<Register*> registerArray;
ALU* alu;
Memory mem;

//0 - 30 are general-purpose registers
//31 - 36 are status registers
//Pointers are used so one array can have both general-purpose and
    specialised registers.
public:
void setRegister(int register, int data); //Set the value of a register
int getRegister(int register); //Get the value of a register
ControlUnit();
~ControlUnit();
vector<string> ReadFile(string);
string FetchNext(bool);
void Decode(string, bool);
int programLength;
std::bitset<32> ConvertToBinary(int val);
int getValueOfArg(string);
int getMemory(int location);

int ValidateInstructionCondition(string PotentialInstruct);
bool ValidCondition(string condition);
bool ValidInstruction(string instruction);
bool checkConditionFlag(std::string condition);
};
```

A.2 ControlUnit.cpp

```
#include "ControlUnit.h"
#include <fstream>
#include <iostream>
using namespace std;

void ControlUnit::setRegister(int registerNumber, int data) {
registerArray[registerNumber]->set(data);
};

int ControlUnit::getRegister(int registerNumber) {
return registerArray[registerNumber]->get();
};
```

```
int ControlUnit::getMemory(int location){
return mem.getMemory(location);
}

ControlUnit::ControlUnit() {
for(int i = 0; i < 37; i++){
if(i == 15){
PC* pc = new PC;
pc->set(0);
registerArray.push_back(pc);
}
else{
registerArray.push_back(new Register);
}
}
alu = new ALU();
programLength = mem.getProgramLength();
};

ControlUnit::~ControlUnit() {
for(vector<Register*>::iterator it = registerArray.begin(); it !=
    registerArray.end(); it++){
delete *it;
}
delete alu;
}

vector<string> ControlUnit::ReadFile(string path) {
string ins;
ifstream myfile(path);
vector<string> lines;

if (myfile.is_open()) {
while (getline(myfile, ins))
lines.push_back(ins);
}
myfile.close();

return lines;
}

void ControlUnit::Decode(string ListOfIns, bool debug) {
string SepIns[4] = {"", "", "", ""};
string op = "";
int result;
SepIns[3] = "none";

//decode
int i = 0;
for (string::iterator it = ListOfIns.begin(); it != ListOfIns.end(); it
    ++) {
if (*it == ' ') {
SepIns[i] = op;
op = "";
i++;
}
else {
```

```
op += *it;
}
}
SepIns[i] = op;

int Args[3] = {0, 0, 0};

for(int i = 1; i < 4; i++){
if(SepIns[i] != "none" && SepIns[i] != ""){
Args[i-1] = getValueOfArg(SepIns[i]);
}
}

//execute
int validationResult = ValidateInstructionCondition(SepIns[0]);
std::string instruction = SepIns[0];
std::string condition = "";
bool conditionResult = false;

switch(validationResult){
case 0:
//both valid
condition = SepIns[0].substr(SepIns[0].length() - 2); // last 2 letters
instruction = SepIns[0].substr(0, SepIns[0].size() - 2); // all but
    last 2 letters
conditionResult = checkConditionFlag(condition);
if(debug) std::cout << "Checking␣condition:␣" << condition << std::endl
    ;
if(!conditionResult){
if(debug) std::cout << "Condition␣returned␣false" << std::endl;
break;
}
if(debug) std::cout << "Condition␣returned␣true" << std::endl;
case 1:
//Instruction valid but not condition
if (instruction == "STR") {
mem.setMemory(Args[0], Args[1]);
}
else if (instruction == "LDR") {
setRegister(Args[0], mem.getMemory(Args[1]));
}
else if(instruction == "MOV"){
setRegister(Args[1], Args[0]);
}
else if (instruction == "B"){
mem.branchTo(SepIns[1], registerArray[15], debug);
}
else {
if(debug) cout << instruction << "␣" << Args[0] << "␣" << Args[1] <<
    endl;
// alu->Control(SepIns[0], stoi(SepIns[1]), stoi(SepIns[2]),
    registerArray[31]);
result = alu->Control(instruction, Args[0], Args[1], registerArray[31])
    ;
if(instruction == "CMP"){
registerArray[31]->set(result);
}
```

```
else{
cout << result << endl;
}
if(SepIns[3] != "none"){
setRegister(Args[2], result);
if(debug) cout << "Stored␣in␣r" << Args[2] << endl;
}
}
break;

case 2:
//Neither valid

break;
}
}

int ControlUnit::getValueOfArg(std::string argument){
if(argument.at(0) == 'R' || argument.at(0) == 'r'){ //Lokesh
string val = argument.substr(1);
int toReturn = 0;
try{
toReturn = stoi(val);
}
catch(exception err){
return 0;
}
toReturn = getRegister(toReturn);
return toReturn;
}
else if(argument.at(0) == '#'){ //Hayden
string val = argument.substr(1);
int toReturn = 0;
try{
toReturn = stoi(val);
}
catch(exception err){
return 0;
}
return toReturn;
}
else{ //Dinkie
std::cout << "ERROR:␣Must␣include␣either␣R␣or␣#␣prefix" << std::endl;
return 0;
}
}

std::bitset<32> ControlUnit::ConvertToBinary(int val)
{
std::bitset<32> BinVal = val; // convert int val to 32 bit binary num
return BinVal;
}

bool ControlUnit::checkConditionFlag(std::string condition){
int iCPSR = registerArray[31]->get(); // get int val of CPSR (R31)
std::bitset<32> bCPSR = ConvertToBinary(iCPSR); // converts int val of
    CPSR to binary value
```

```
//31 - N - Negative
//30 - Z - Zero
//29 - C - Carry
//28 - V - Overflow

if(condition == "NE"){ //Not Equal
if(!bCPSR[30]) return true;
}
else if(condition == "EQ"){ //Equal
if(bCPSR[30]) return true;
}
else if(condition == "CS"){ //Carry set
if(bCPSR[29]) return true;
}
else if(condition == "CC"){ //Carry unset
if(!bCPSR[29]) return true;
}
else if(condition == "MI"){ //Minus/negative
if(bCPSR[31]) return true;
}
else if(condition == "PL"){ //Positive/zero
if(!bCPSR[31]) return true;
}
else if(condition == "VS"){ //Overflow set
if(bCPSR[28]) return true;
}
else if(condition == "VC"){ //Overflow unset
if(!bCPSR[28]) return true;
}
else if(condition == "HI"){ //Unsigned higher
if(bCPSR[29] && !bCPSR[30]) return true;
}
else if(condition == "LS"){ //Unsigned lower or equal
if(bCPSR[30] && !bCPSR[29]) return true;
}
else if(condition == "GE"){ //Signed greater than or equal
if(bCPSR[31] == bCPSR[28]) return true;
}
else if(condition == "LT"){ //Signed less than
if(bCPSR[31] != bCPSR[28]) return true;
}
else if(condition == "GT"){ //Signed greater than
if(!bCPSR[30] && (bCPSR[31] == bCPSR[28])) return true;
}
else if(condition == "LE"){ //Signed less than or equal
if(bCPSR[30] && (bCPSR[31] != bCPSR[28])) return true;
}
else if(condition == "AL"){ //Unconditional
return true;
}
return false;
}

string ControlUnit::FetchNext(bool debug){
string nextInstruction = mem.getNextInstruction(registerArray[15]->get
    (), debug);
```

```
registerArray[15]->increment();
return nextInstruction;
}

bool ControlUnit::ValidCondition(string condition)
{
//list of all valid conditions for ARM7TDMI (According to technical
    reference manual)
vector<string> ValidConditions = {"EQ", "NE", "CS", "CC", "MI", "PL", "
    VS", "VC", "HI", "LS", "GE", "LT", "GT", "LE", "AL"};

//Iterate through valid conditions
for (int i = 0; i < ValidConditions.size(); i++)
{
if (condition == ValidConditions[i]) // if match found
{
return true; // condition is valid, so return true
}
}
return false;
}

bool ControlUnit::ValidInstruction(string instruction)
{
//list of all valid ARM Instructions for ARM7TDMI (According to
    technical reference manual)
vector<string> ValidInstructions = { "MOV", "SUB", "ADD", "MUL", "CMP",
     "AND", "EOR", "ORR", "B", "BL", "BX", "LDR", "STR",/*  "LDC", "MVN
    ", "MRS", "MSR",  "ADC" , "SBC", "RSB" */};

//Iterate through valid instructions
for (int i = 0; i < ValidInstructions.size(); i++)
{
if (instruction == ValidInstructions[i]) // if match found
{
return true; // instruction is valid, so return true
}
}
return false;
}

int ControlUnit::ValidateInstructionCondition(string CondAndInstr)
{
string condition = CondAndInstr.substr(CondAndInstr.length() - 2); //
    last 2 letters
string Instruction = CondAndInstr.substr(0, CondAndInstr.size() - 2);
    // all but last 2 letters

bool ValidCond = ValidCondition(condition); // true if valid condition,
     false if invalid
if(!ValidCond){
Instruction = CondAndInstr;
}
bool ValidInstr = ValidInstruction(Instruction);

if (ValidCond && ValidInstr)
{
```

```
// check condition then run instruction if true
return 0;
}
else if (!ValidCond && ValidInstr)
{
// run instruction only
return 1;
}
else
{
// don't run anything
return 2;
}

}
```

A.3 ALU.h

```
#pragma once
#include <iostream>
#include <bitset>
#include <limits.h>
#include "Register.h"

class ALU {
public:
int Control(std::string Opcode, int Operand1, int Operand2, Register*
    CPSR); // Decodes opcode and conducts ALU functions (e.g. ADD, MUL
    etc.)
private:

//Core ALU Functions
int ADD(int Operand1, int Operand2); // adds 2 operands, returns result
int SUB(int Operand1, int Operand2); // subtracts 2 operands, returns
    result
int MUL(int Operand1, int Operand2); // multiplies 2 operands, returns
    result
int AND(int Operand1, int Operand2); // bitwise AND comparison, returns
     result
int EOR(int Operand1, int Operand2); // bitwise XOR comparison, returns
     result
int ORR(int Operand1, int Operand2); // bitwise OR comparison, returns
    result
int CMP(int Operand1, int Operand2, Register* CPSR); // Compare equal
    to, updates zero flag of program status register (True if equal,
    False if not equal)


//Other Functions needed for Core ALU Functions (e.g. Abstraction of
    validation & binary conversions)
std::bitset<32> ConvertToBinary(int val);
void ValidateADD(Register* CPSR, int Operand1, int Operand2);
void ValidateSUB(Register* CPSR, int Operand1, int Operand2);
void ValidateMUL(Register* CPSR, int Operand1, int Operand2);

};
```

A.4 ALU.cpp

```
#pragma once
#include "ALU.h"
#include "register.h"

//Public:
//Control Function
int ALU::Control(std::string Opcode, int Operand1, int Operand2,
    Register* CPSR)
{
int ReturnVal = 0;
if (Opcode == "ADD")
{
ValidateADD(CPSR, Operand1, Operand2);
ReturnVal = ADD(Operand1, Operand2);
}
else if (Opcode == "SUB")
{
ValidateSUB(CPSR, Operand1, Operand2);
ReturnVal = SUB(Operand1, Operand2);
}
else if (Opcode == "MUL")
{
ValidateMUL(CPSR, Operand1, Operand2);
ReturnVal = MUL(Operand1, Operand2);
}
else if (Opcode == "AND")
{
ReturnVal = AND(Operand1, Operand2);
}
else if (Opcode == "EOR")
{
ReturnVal = EOR(Operand1, Operand2);
}
else if (Opcode == "ORR")
{
ReturnVal = ORR(Operand1, Operand2);
}
else if (Opcode == "CMP")
{
ReturnVal = CMP(Operand1, Operand2, CPSR);
}
return ReturnVal;
}


//Private:
//Core ALU Functions
int ALU::ADD(int Operand1, int Operand2)
{
int result = 0;
result = Operand1 + Operand2;
return result;
}

int ALU::SUB(int Operand1, int Operand2)
```

```
{
int result = 0;
result = Operand1 - Operand2;
return result;
}

int ALU::MUL(int Operand1, int Operand2)
{
int result = 0;
result = Operand1 * Operand2;
return result;
}

int ALU::AND(int Operand1, int Operand2)
{
std::bitset<32> bOperand1 = ConvertToBinary(Operand1); // convert int
    operands to binary operands
std::bitset<32> bOperand2 = ConvertToBinary(Operand2);

std::bitset<32> bResult = bOperand1 &= bOperand2; // perform AND
    operation & store in bResult

int iResult = (int)(bResult.to_ulong()); // convert bResult to int
    iResult
return iResult;
}

int ALU::EOR(int Operand1, int Operand2)
{
std::bitset<32> bOperand1 = ConvertToBinary(Operand1); // convert int
    operands to binary operands
std::bitset<32> bOperand2 = ConvertToBinary(Operand2);

std::bitset<32> bResult = bOperand1 ^= bOperand2; // perform XOR
    operation & store in bResult

int iResult = (int)(bResult.to_ulong()); // convert bResult to int
    iResult
return iResult;
}

int ALU::ORR(int Operand1, int Operand2)
{
std::bitset<32> bOperand1 = ConvertToBinary(Operand1); // convert int
    operands to binary operands
std::bitset<32> bOperand2 = ConvertToBinary(Operand2);

std::bitset<32> bResult = bOperand1 |= bOperand2; // perform OR
    operation & store in BinResult

int iResult = (int)(bResult.to_ulong()); // convert BinResult to int
    iResult
return iResult;
}

int ALU::CMP(int Operand1, int Operand2, Register* CPSR)
{
```

```
int iCPSR = CPSR->get(); // get int val of CPSR (R31)
std::bitset<32> bCPSR = ConvertToBinary(iCPSR); // converts int val of
    CPSR to binary value

ValidateSUB(CPSR, Operand1, Operand2);
int difference = SUB(Operand1, Operand2); // gets difference of operand
     1 or 2
if (difference == 0) //if difference is 0, operands are equal
{
bCPSR.set(30, true); // change 30th bit (zero flag) of CPSR to True
}
else // operands are not equal
{
bCPSR.set(30, false); // change 30th bit (zero flag) of CPSR to False
}

iCPSR = (int)(bCPSR.to_ulong()); // convert back to int
CPSR->set(iCPSR); // set CPSR to new val
return iCPSR;
}

//Other Functions needed for Core ALU Functions (e.g. Abstraction of
    validation & binary conversions)

std::bitset<32> ALU::ConvertToBinary(int val)
{
std::bitset<32> BinVal = val; // convert int val to 32 bit binary num
return BinVal;
}

void ALU::ValidateADD(Register* CPSR, int Operand1, int Operand2)
{
int iCPSR = CPSR->get(); // get int val of CPSR (R31)
std::bitset<32> bCPSR = ConvertToBinary(iCPSR); // converts int val of
    CPSR to binary value




if (((Operand2 > 0) && (Operand1 > (INT_MAX - Operand2))) || ((Operand2
     < 0) && (Operand1 < (INT_MIN - Operand2))))
{
bCPSR.set(28, true); // set overflow flag to TRUE
}


iCPSR = (int)(bCPSR.to_ulong()); // convert back to int
CPSR->set(iCPSR); // set CPSR to new value
}

void ALU::ValidateSUB(Register* CPSR, int Operand1, int Operand2)
{
int iCPSR = CPSR->get(); // get int val of CPSR (R31)
std::bitset<32> bCPSR = ConvertToBinary(iCPSR); // converts int val of
    CPSR to binary value

int MinIntSize = -2147483647;
```

```
if ((Operand2 > 0 && Operand1 < INT_MIN + Operand2) || (Operand2 < 0 &&
     Operand1 > INT_MAX + Operand2)) {
bCPSR.set(28, true); // set overflow flag to TRUE
}


iCPSR = (int)(bCPSR.to_ulong()); // convert back to int
CPSR->set(iCPSR); // set CPSR to new value
}

void ALU::ValidateMUL(Register* CPSR, int Operand1, int Operand2)
{
int iCPSR = CPSR->get(); // get int val of CPSR (R31)
std::bitset<32> bCPSR = ConvertToBinary(iCPSR); // converts int val of
    CPSR to binary value

long long int LLIoperand1 = (long long int)Operand1;
long long int LLIoperand2 = (long long int)Operand2;

long long int Temp = LLIoperand1 * LLIoperand2; // check calculation
    and store in Temp

if ((Temp > INT_MAX) || (Temp < INT_MIN)) // if temp is too large or
    small
{
bCPSR.set(28, true); // set overflow flag to TRUE
}

iCPSR = (int)(bCPSR.to_ulong()); // convert back to int
CPSR->set(iCPSR); // set CPSR to new value
}
```

A.5 memory.h

```
#pragma once
#include <string>
#include "register.h"
#include <vector>

class Memory{
private:
std::vector<std::string> program; //Program is a vector of strings
std::vector<int> mem; //Memory is a vector of ints
int programLength;
public:
Memory();
~Memory();
int getProgramLength();
std::string getNextInstruction(int pc, bool);
bool branchTo(std::string label, Register* pc, bool);
int getMemory(int location);
void setMemory(int location, int value);
};
```

A.6 memory.cpp

```
#pragma once
#include "memory.h"
#include <iostream>
#include <fstream>
#include <vector>

Memory::Memory(){ //Constructor for Memory object
std::ifstream file("input.txt"); //Input file is "./input.txt"
std::cout << "Read␣file" << std::endl;
if(!file){ //If file doesn't exist, throw error
std::cout << "ERROR:␣Input␣file␣not␣valid" << std::endl;
}
else{
std::string next;
while(std::getline(file, next)){
program.push_back(next); //Push instruction
programLength++;
}
}
file.close(); //Close file after use

for(int i = 0; i < 1024; i++){
mem.push_back(0);
}
};

int Memory::getProgramLength(){
return programLength;
}

Memory::~Memory(){
delete &mem; //Prevent memory leak
delete &program;
};

std::string Memory::getNextInstruction(int pc, bool debug){
if(debug) std::cout << "Getting␣instruction␣at␣" << pc << std::endl;
std::string nextInstruction = program.at(pc); //Next instruction to
    execute is at position stored at PC
return nextInstruction;
};

int Memory::getMemory(int location) {
return mem[location];
};

void Memory::setMemory(int location, int value) {
mem[location] = value;
};

bool Memory::branchTo(std::string label, Register* pc, bool debug){
if(debug) std::cout << "Searching␣for␣label:␣" << label << std::endl;
int index = 0;
for(auto it = program.begin(); it != program.end(); it++){
if(*it == label){
```

```
pc->set(index+1);
if(debug) std::cout << "Label␣found␣at␣" << index << std::endl;
return true;
}
index++;
}
return false;
};
```

A.7 Register.h

```
#pragma once
class Register {
protected:
int data; //The value of this register
public:
int get(); //Get the value of this register
void set(int data); //Set the value of this register
virtual void increment();
virtual void increment(int amount);
Register();
};

class PC : public Register{
public:
virtual void increment();
virtual void increment(int amount);
};
```

A.8 Register.cpp

```
#include "register.h"
#include <iostream>

Register::Register(){
data = 0;
}

int Register::get() {
return data;
};

void Register::set(int data) {
this->data = data;
};

void Register::increment(){
std::cout << "ERROR:␣Can't␣increment␣this␣register" << std::endl;
}

void Register::increment(int amount){
std::cout << "ERROR:␣Can't␣increment␣this␣register" << std::endl;
}

void PC::increment(){
data++; //Default increment by one
```

```
};

void PC::increment(int amount){
data += amount; //Useful for "skip next x instructions if..."
};
```

A.9 maintTest.cpp

```
#pragma once
#include "ALU.h"
#include "ControlUnit.h"
#include "memory.h"
#include <fstream>

int main(int arg){
ControlUnit* cu = new ControlUnit();
// ALU alu;
// std::cout << alu.Control("ADD", 1, 2, new Register) << std::endl;

bool debug = true;
if(arg == 1){
debug = false;
}

std::ofstream registerLog;
std::ofstream memoryLog;
std::ofstream assemblyLog;

registerLog.open("registerLog.log", std::ofstream::out | std::ofstream
    ::trunc);
memoryLog.open("memoryLog.log", std::ofstream::out | std::ofstream::
    trunc);
assemblyLog.open("assemblyLog.log", std::ofstream::out | std::ofstream
    ::trunc);

for(int i = 0; i < cu->programLength; i++){
assemblyLog << cu->FetchNext(true) << std::endl;
}

cu->setRegister(15, 0);

for(int i = 0; cu->getRegister(15) < cu->programLength; i++){
std::string nextInstruction = cu->FetchNext(debug);
cu->Decode(nextInstruction, debug);
for(int ii = 0; ii < 1024; ii++){
memoryLog << cu->getMemory(ii) << "␣";
}
for(int ii = 0; ii < 37; ii++){
if(ii != 31){
registerLog << cu->getRegister(ii) << "␣";
}
else{
std::bitset<32> PSR = cu->getRegister(ii);
registerLog << PSR << "␣";
}
}
memoryLog << std::endl;
```

```
registerLog << std::endl;
}

return 0;
}
```

A.10 index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Reverse Debugger</title>
<link rel="stylesheet" href="styles.css" type="text/css">
<script>require('./functions.js');</script>
</head>

<body>
<h1>ARM7-TDMI Reverse Debugger</h1>
<div id="memory">
</div>
<div id="registers">
</div>
<div id="assembly">
</div>
<div id="controls">
<button type="button" data-value=-5 id="back5">Back 5</button>
<button type="button" data-value=-1 id="back">Back</button>
<button type="button" data-value=1 id="step">Step</button>
<button type="button" data-value=5 id="step5">Step 5</button>
<button type="button" data-value=10 id="step10">Step 10</button>
</div>
</body>
</html>
```

A.11 main.js

```
const { app, BrowserWindow} = require('electron');
var window;

function createWindow () {
window = new BrowserWindow({ width: 1280, height: 720});
window.loadFile('index.html');
window.on('closed', () => {
window = null;
});
}

app.on('ready', createWindow);

app.on('window-all-closed', () => {
if (process.platform !== 'darwin') {
app.quit();
}
});

app.on('activate', () => {
```

```
if (window === null) {
createWindow();
}
});
```

A.12 functions.js

```
const fs = require("fs");
var assemblyLog;
var memoryLog;
var registerLog;
var index = 0;
var buttons;

window.onload = () => {
readLogs();
document.getElementById("back5").onclick = function(){
index += parseInt(document.getElementById("back5").dataset.value);
writeLogs(index);
}
document.getElementById("back").onclick = function(){
index += parseInt(document.getElementById("back").dataset.value);
writeLogs(index);
}
document.getElementById("step").onclick = function(){
index += parseInt(document.getElementById("step").dataset.value);
writeLogs(index);
}
document.getElementById("step5").onclick = function(){
index += parseInt(document.getElementById("step5").dataset.value);
writeLogs(index);
}
document.getElementById("step10").onclick = function(){
index += parseInt(document.getElementById("step10").dataset.value);
writeLogs(index);
}
}

async function readLogs(){
memoryLog = await fs.readFileSync("..\\memoryLog.log").toString().split
    ("\n");
registerLog = await fs.readFileSync("..\\registerLog.log").toString().
    split("\n");
assemblyLog = await fs.readFileSync("..\\assemblyLog.log").toString().
    split("\n");

splitLogs();
}

function splitLogs(){
for(var i = 0; i < memoryLog.length; i++){
memoryLog[i] = memoryLog[i].split("␣");
}
for(var i = 0; i < registerLog.length; i++){
registerLog[i] = registerLog[i].split("␣");
}
drawLogs();
```

```
}

function drawLogs(){
for(var i = 0; i < memoryLog[0].length; i++){
document.getElementById("memory").innerHTML += `<div id=mem${i}></div
    >`;
}
for(var i = 0; i < registerLog[0].length - 1; i++){
document.getElementById("registers").innerHTML += `<div id=reg${i}></
    div>`;
}
for(var i = 0; i < assemblyLog.length; i++){
document.getElementById("assembly").innerHTML += `<div id=asm${i}></div
    >`;
}
writeLogs(0);
}

function writeLogs(index){
console.log(index);
if(index < 0){
index = 0;
}
else if(index >= memoryLog.length){
index = memoryLog.length-1;
}
for(var i = 0; i < memoryLog[index].length; i++){
document.getElementById(`mem${i}`).innerHTML = memoryLog[index][i];
}
for(var i = 0; i < registerLog[index].length - 1; i++){
document.getElementById(`reg${i}`).innerHTML = registerLog[index][i];
}
for(var i = 0; i < assemblyLog.length; i++){
document.getElementById(`asm${i}`).innerHTML = assemblyLog[i];
if(i == parseInt(document.getElementById("reg15").innerHTML)-1){
document.getElementById(`asm${i}`).style.background = "#7289DA";
document.getElementById(`asm${i}`).style.color = "#FFFFFF";
}
else{
document.getElementById(`asm${i}`).style.background = "none";
document.getElementById(`asm${i}`).style.color = "#99AAB5";
}
}
}
```

A.13 style.css

```
body{
padding: 0;
margin: 0;
background-color: #23272A;
}

html{
padding: 0;
margin: 0;
}
```

```
h1{
padding: 0;
margin: 0;
margin-left: 2.5vw;
margin-top: 2.5vh;
height: 5vh;
color: #99AAB5;
font-family: monospace;
}

#memory{
margin-top: 2.5vh;
float: left;
height: 87.5vh;
width: 46.25vw;
margin-right: 2.5vw;
margin-left: 2.5vw;
background-color: #2C2F33;
display: flex;
flex-flow: wrap;
align-content: space-evenly;
overflow-y: scroll;
grid-template-columns: auto auto auto auto auto auto auto auto auto
    auto auto auto auto auto auto auto auto auto auto
    auto auto auto auto auto auto auto auto auto auto;
grid-template-rows: auto auto auto auto auto auto auto auto auto auto
    auto auto auto auto auto auto auto auto auto auto
    auto auto auto auto auto auto auto auto auto auto;
overflow-y: scroll;
}

#registers{
margin-top: 2.5vh;
float: right;
height: 20vh;
width: 46.25vw;
margin-right: 2.5vw;
background-color: #2C2F33;
display: flex;
flex-flow: wrap;
align-content: space-evenly;
overflow-y: scroll;
}

#controls{
background-color: #2C2F33;
float: right;
margin-top: 2.5vh;
height: 65vh;
width: 21.875vw;
}

#controls button{
height: 20%;
width: 100%;
}
```

```
#assembly{
float: right;
height: 65vh;
width: 21.875vw;
margin-top: 2.5vh;
margin-left: 2.5vw;
margin-right: 2.5vw;
background-color: #2C2F33;
display: grid;
grid-template-columns: auto;
overflow-y: scroll;
justify-content: space-evenly;
align-content: space-evenly;
}

#memory div{
padding: 2px;
color: #99AAB5;
}

#reg15{
background-color: #551111 !important;
}

#reg31{
background-color: #000055 !important;
}

#registers div{
padding: 5px;
margin-left: 1%;
border-radius: 2.5px;
background-color: #23272A;
color: #99AAB5;
}

#assembly div{
color: #99AAB5;
display: block;
float: left;
padding: 5px;
border-radius: 2.5px;
}
```

A.14 input.txt

```
MOV #1 #0
MOV #1 #1
MOV #2 #3
STR #0 #1
STR #1 #1
MOV #25 #5
CMP R4 R5
#start
STR R4 R0
ADD R0 R1 #3
```

```
MOV R1 #0
MOV R3 #1
ADD #1 R4 #4
CMP R4 R5
BNE #start
```

B Record and Summary of Meetings

Below are listed all meetings that took place as a part of this project, along with a brief summary of what was discussed and any key decisions that were made. Unless otherwise stated, all group members were in attendance.

B.1 08/02/2019

Decided to emulate an ARM architecture, will research and choose a specific chip by Monday and move on to writing proposal. Decided to always have a demo ready at any point of the project, using Version Control on GitHub.

B.2 11/02/2019

Decided on ARM7TDMI, split the proposal up into sections, meeting again 11:30 on Friday with supervisor. Planned to finish proposal over weekend.

B.3 15/02/2019

Breaking down plan into core and optional objectives, writing register class and finishing report over weekend make any change on Monday and sent to supervisor by Tuesday.
Sam and Lokesh did not attend

B.4 18/02/2019

Finished proposal, wrote up plan, finished aims & drew UML diagram of program structure. Plan to send final proposal to supervisor later in the day.

B.5 01/03/2019

20 minute meeting, all happy, need to work on communication... Demo due next week.
Alex, Sam and Lokesh did not attend

B.6 04/03/2019

20 minute meeting, stressed importance of attendance. Meeting again Thursday 1pm to finalise core elements assigned to each team. Went over use of github.
Lokesh did not attend

B.7 07/03/2019

2hr meeting - Merged code (with some issues), Dinkie is making demo for tomorrow, Tasks handed out

B.8 08/03/2019

20 minute meeting, showed Charles progress so far, everyone's happy Need to think about how we're going to show the project off, maybe a log file and a separate program that displays it visually? Everyone has assigned tasks from yesterday Meeting again Monday at 4

B.9 11/03/2019

Split off into pairs to discuss work packages:
Alex and Sam - 45 minute meeting
Hayden and Lokesh - 60 minute meeting
Matt and Dinkie - 60 minute meeting

B.10 15/03/2019

20 minute meeting Charles is happy with load/store, recommended using Latex for second report, decided on reverse debugger (electron?) Branching and addressing modes working for next meeting with Charles Matt didn't attend

B.11 22/03/2019

45 minute meeting Addressing modes given an extra week, Dinkie assigned along with Lokesh and Hayden First draft report/Latex use, Matt Think about demo codes (Fibonacci?) Logging of each step in the program (registers at each point, memory, pc stack, next instruction), Sam, Alex and Hayden did not attend @Hayden @lokesh Meeting on Monday at 4 to go over addressing modes, if it doesn't work by Wednesday, throw more people at it until it does

B.12 29/03/2019

Margin notes in latex, focus on ticking boxes in the CRG Agile vs waterfall etc. Agile has worked REALLY well for us in terms of needing extra time for addressing mode and reverse debugger at the end Write a tutorial for the emulator that the examiner can follow and use the system for dem Find existing code/tutorials for ARM programming, can they run on our emulator? Recursion possible?

50 minute meeting Sam not here

B.13 05/04/2019

40 minute meeting - Charles is happy still, Need to send Charles final report on Tuesday, everyone has bits allocated

C Solution Demonstration

Figure 2: State of program after "MOV #25 #5" is run

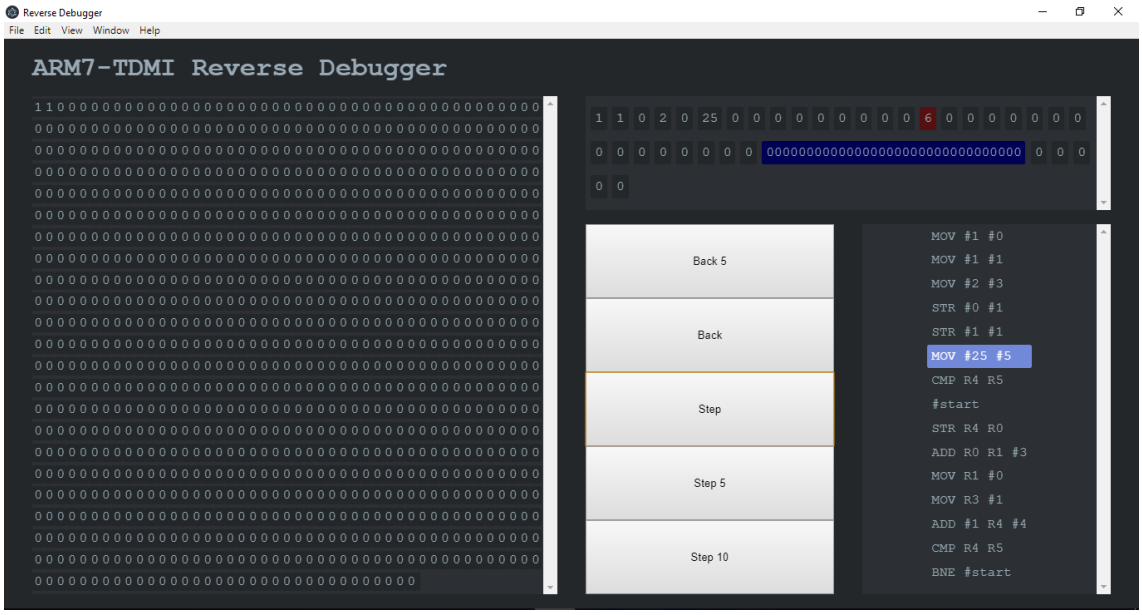


Figure 3: State of program before "MOV #25 #5" is run

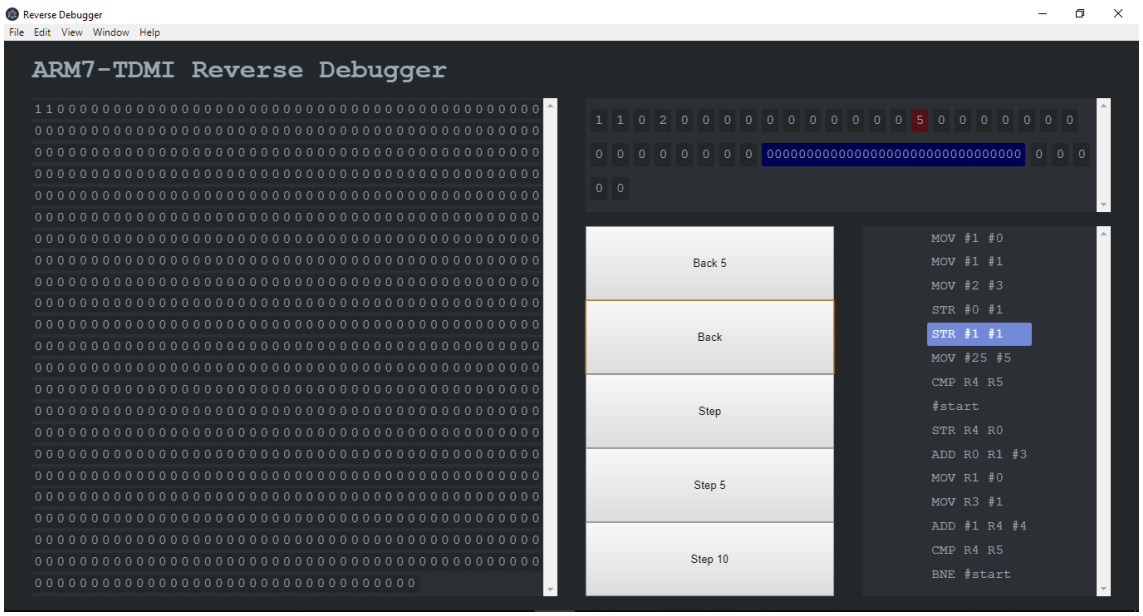


Figure 4: Debugger showing the state of registers

Figure 5: Debugger showing the state of memory, the user can click to highlight positions to make keeping track of them easier

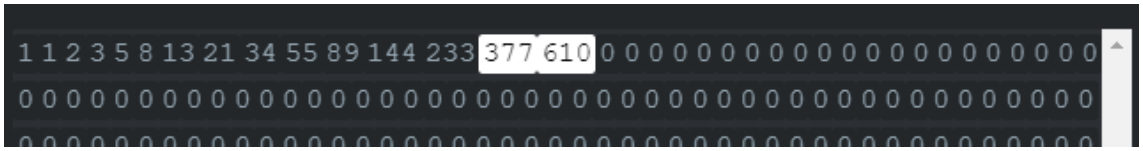


Figure 6: Output in debug mode during execution of the Fibonacci example code, showing branch, add and store instructions in use

```
Getting instruction at 12
ADD 1 22
23
Stored in r4
Getting instruction at 13
CMP 23 25
Getting instruction at 14
Checking condition: NE
Condition returned true
Searching for label: #start
Label found at 7
Getting instruction at 8
Getting instruction at 9
ADD 46368 75025
121393
Stored in r3
Getting instruction at 10
Getting instruction at 11
Getting instruction at 12
ADD 1 23
24
Stored in r4
```

Figure 7: Fibonacci example code

```
0: MOV #1 #0
1: MOV #1 #1
2: MOV #2 #3
3: STR #0 #1
4: STR #1 #1
5: MOV #25 #5
6: CMP R4 R5
7: #start
8: STR R4 R0
9: ADD R0 R1 #3
10: MOV R1 #0
11: MOV R3 #1
12: ADD #1 R4 #4
13: CMP R4 R5
14: BNE #start
```

Figure 8: When hovering over a position in memory, the registers or the assembly code, the index of that position is displayed in the top right corner

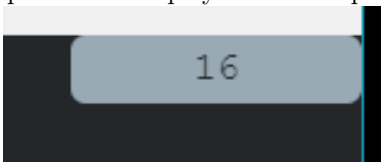


Figure 9: Users can step through the code in different ways, including going forward by either 1, 5 or 10 instructions, or backwards by 1 or 5 instructions at a time

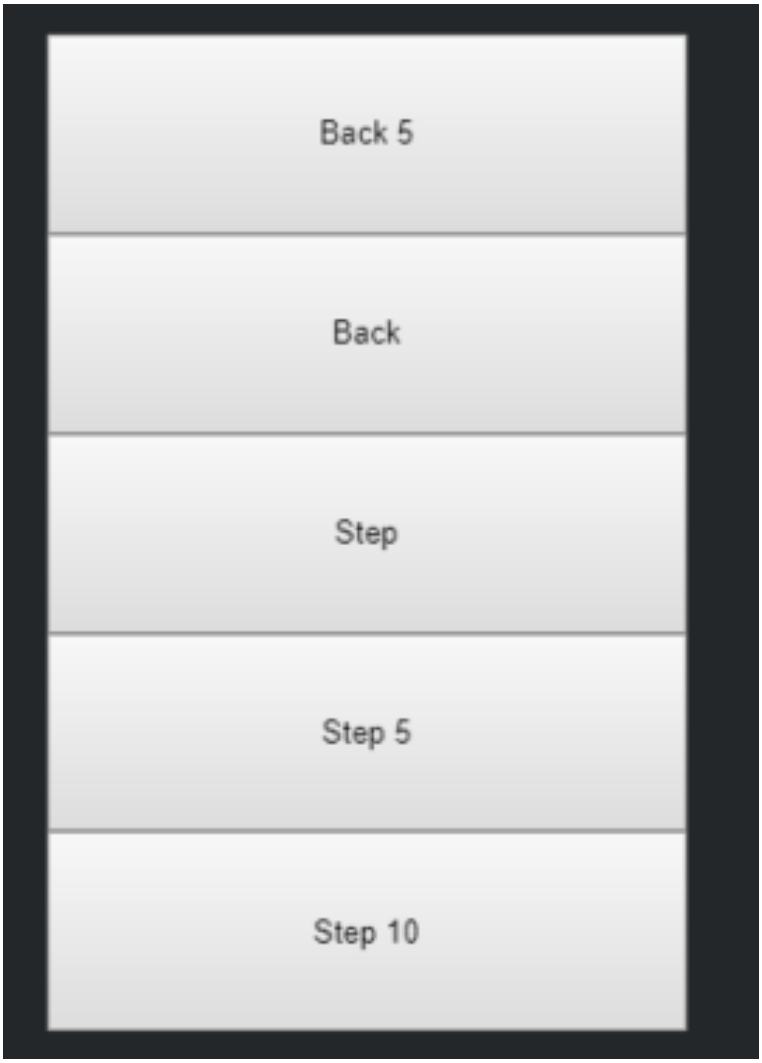


Figure 10: The status register (highlighted in blue) has different bits set after compare instructions are run. In this case, "CMP #25 #25" was run which set the Z (zero) flag to true

Figure 11: The program counter (highlighted in red) stores the current position in the code. It can be modified either directly or through branch instructions to execute subroutines or loops. The Fibonacci example code uses a for loop, incrementing register 4 and comparing it to the value in register 5