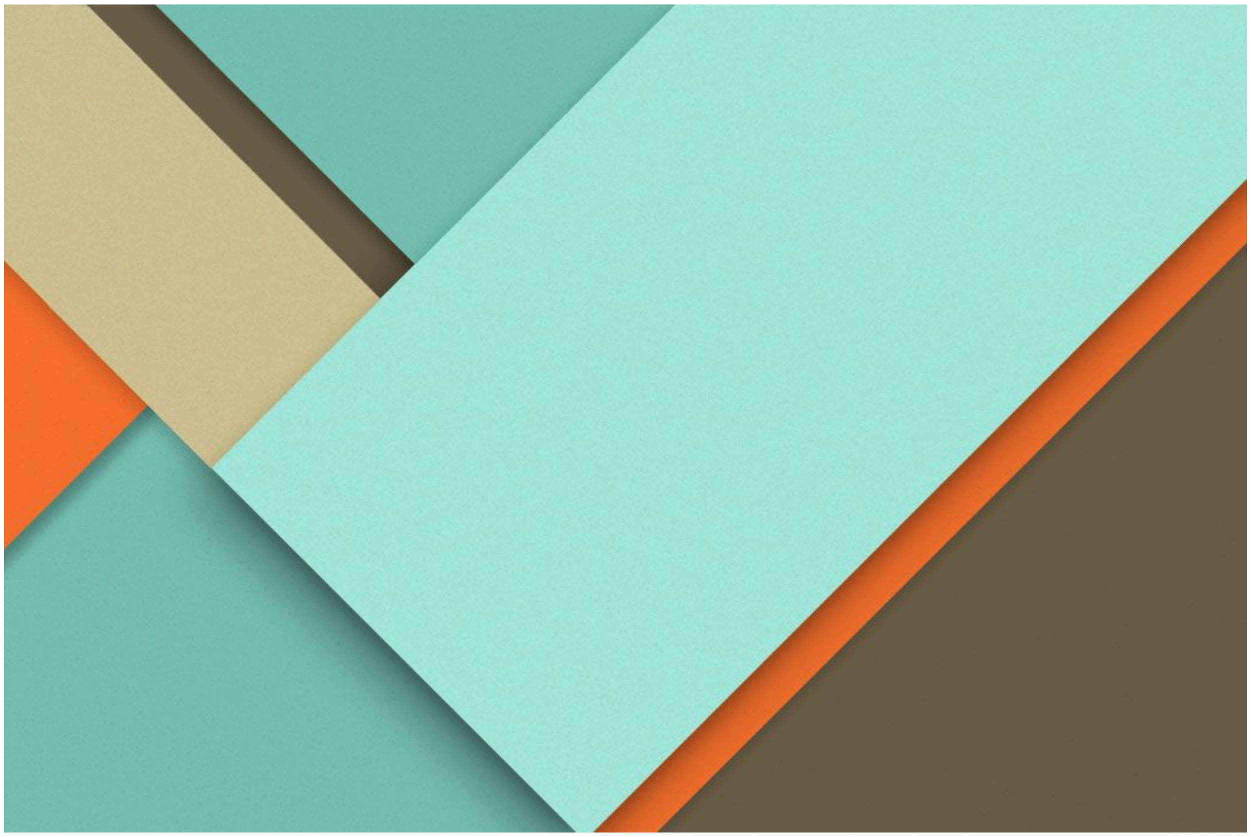

ARM Simulator

— in JAVA

TEAM MEMBERS

Pratyush Kaushal	2016070
Adarsh Kumar Choudhary	2016007
Sushant Kumar Singh	2016103

Overview

Designing and implementing a function simulator in JAVA for subset of ARM instructions.

The ARM Simulator (written in JAVA) will take as input/reads a MEM file (*.mem) and would then execute the instructions as though they were executed on a system with ARM architecture. This would enable simulating ARM code on any system which has JVM installed on it (and hence can run JAVA code), irrespective of the architecture of the system.

The Simulator will take instructions (from MEM file) one at a time and execute as per its functional behavior and prints the messages for each stage. The Simulator stops the simulation once the instruction sequence corresponds to "SWI 0x11" in ARM and exits.

Stages for the instruction sequences are as follows:

- ➔ **Fetch**
- ➔ **Decode**
- ➔ **Execute**
- ➔ **Memory**
- ➔ **Write-Back**

Methodology

Following is the program flow showing how the Simulation will be done given a MEM file as input.

1. **Initialise: The Registers R0-R15 are initialised to 0, Instruction Memory is freed (each byte initialised to 0x00000000) and other variables such as flags, operands, etc are also reset to 0.**
2. **Load Instruction Memory: Read the MEM file and load the Instruction Memory .**
3. **Run the Simulation:**
    - ➔ **An infinite loop runs**
    - ➔ **In each iteration functions fetch, decode, execute, memory, write-back are called consecutively where each steps performs accordingly and prints the message with details of the action performed .**
    - ➔ **The execution stops if decode stage finds the instruction corresponding to "SWI 0x11" and the program quits.**

The functions perform as follow:

1. FETCH: Read the next instruction from Instruction Memory based on program counter and load the instruction register.
2. DECODE: Decode the instruction and decide which type of instruction it is and what operation to performed by Execute stage. Find the operands. Read the registers corresponding to operand register for execute stage.
3. EXECUTE: Perform the ALU operations based on control signals defined in decode stage.
4. MEMORY: Do the operations related to memory. In case of load and store instruction, load the data from memory or store the data to memory accordingly.
5. WRITEBACK: Write the result to the destination register, defined from decode stage.

Project Plan

As we are familiar with JAVA Data Structures and Algorithms we have decided to implement the ARM Simulator using JAVA.

We will use JAVA data structures (Array or HashMap) to define the ARM register files (R0-R15) and Instruction Memory. We will use some extra variables for various flags, program counter, operands, destination register, opcode, control signals, etc so as to make the simulation or program execution easier in JAVA.

1. In initialisation step all the variables defined would be reset/initialised to 0
2. Instruction memory(java array or HashMap) is initialised after reading the given MEM file .
3. We will then take instruction one at a time and perform the execution as outlined in Methodology. Details of the functions called in Simulation step (Step-2 of Methodology) are as followed.

Following Functions called in Simulation step perform the actions and print the message for the same:

1. FETCH: Read the instruction from instruction memory from address value defined by program counter. We may use HashMaps or JAVA arrays to get the instruction from instruction memory based on program counter. Increment the program counter by 4 . Also update the variable that stores the current instruction for ease of use in other functions.

2. DECODE: All ARM Instruction are 32 bits. The significance of bits are as follows:

   (Source: Morgan Kaufmann :Computer Organization and Design)

Cond	F	I	Opcode	S	Rn	Rd	Operand2
4 bits	2 bits	1 bits	4 bits	1 bits	4 bits	4 bits	12 bits

Cond	Related to condition branch instructions
F	Instruction Format.
I	Immediate. If I is 0 , the second source operand to register. If I is 1 the second operand is 12-bit immediate. (4 bit rotate + 8 bit immediate)
Opcode	Basic operation of the instruction
S	Set condition Code (Kind of Conditional Branch Instructions)
Rn	The First Register source operand.
Rd	The Register destination operand. It gets the result of the operation.
Operand2	The second Source operand.

Based on this, this stage then determines what type of instruction is the current instruction (F=0:Data Processing, F=1: Data Transfer, F=2: Branch Instructions) and reads the operand registers for execute stage accordingly.

3. **EXECUTE: Perform the ALU operations based on opcode, F, I, Rn, Operand2 .**
4. **MEMORY: Do the operations related to memory. In case of load and store instruction, load the data from memory or store the data to memory accordingly.**
5. **WRITEBACK: Write the result to the destination register (Rd), defined from decode stage.**

(We may finally print the values stored in memory at last by printing the values stored in memory variables)

After implementation of all the methods we would then test and debug the program.

Timeline

Week 1	Week 2	Week 3	Week 4
1. Create the Basic Structure of Project (method for reading input files, main method, etc.) 2. Init, Simulate method implementation	1. Implement Fetch, Decode method.	1. Implement Execute, Memory,Writeback method.	1. Testing and Debugging

Final Implementation:

- Array of Long is used for Memory, Register files
- Long, Integer data types are used for various Flags, intermediate control and datatpath variables
- getImmediateValue() method is added to make calculation of immediate operand easy

Supported Instructions by ARMSim JAVA:

- AND, EOR, SUB, ADD, ADC, CMP, ORR, MOV, MVN
- LDR, STR with Pre-Post Indexing, Up-Down, Immediate offset, Register offset
- B, BL with EQ, NE, GE, LT, GT, LE, AL
- Swi 0x00, swi 0x11, swi 0x6b, swi 0x6c
- Stack
- function call supported

Contribution:

Aadarsh Kumar Choudhary	execute(), writeBack(), swi_exit(), javaDoc,GUI
Pratyush Kaushal	main(), simulate(), init(), loadMem(), writeMem(), fetch(), getMemoryStatus()
Sushant Kumar Singh	decode(), memory(), getImmediateValue(), javaDoc,GUI

Extension :

- We have Implemented a GUI interface .
- Functionality :
    - Run, Reload ,Stack View , Register View , Step Into ,File Selection Stop and Console.