30 October 2017

Arm Simulator

Computer Organisation Project

Team Members

- Sneha Sinha 2016098
- Satyam Verma 2016087
- Utkarsh Khokhar 2016110

Overview & Purpose

Our project topic is ARM simulator. We will be building a simulator that simulates the manner in which the ARM assembly language is executed. It will have the basic functionalities and support primary operations. We also plan to implement heap functions using SWI as an additional feature.

Language

Java

Concept

ARM assembly code is executed in several stages which are Fetch, Decode, Execute, Memory, and Write-Back. The stages are basically designed as a pipelining interface such that the instruction have stages and if any instruction's any stage is

complete then the resources for that stage could be used for the instruction following it. The detailed functions of each stage and how are we planning to implement it our project is described below in the methodology.

Timeline

Week	Duration	Task	Assigned
Week 1	31/10-1/11	Helper Functions	Sneha
Week 1	31/10-1/11	Fetch	Utkarsh
Week 1	2/11-5/11	Decode	Sneha, Satyam
Week 2	6/11-8/11	Execute	Satyam
Week 2	8/11-11/11	Memory	Utkarsh, Sneha
Week 3	11/11-13/11	Writeback	Satyam
Week 3	14/11-18/11	Testing, Debugging	Sneha,Satyam, Utkarsh

Project Plan and Methodology

The input file consists of the address and the instruction in the form of hex strings which need to be decoded. The address is where the instruction is stored in the memory.

Firstly, the memory is loaded with the input MEM file. For this, we will be using file reader class methods in Java. The method loadmem will include an exception when the file can't be found.

Fetch

This method will read the instruction from the memory(array of strings) and update the instruction register being used by incrementing it by 4.

The simulator will also print a message on the console each time fetch is called, for example: “FETCH: Fetch instruction 0xE3A0200A from address 0x0”

Decode

The ARM instructions include branch instructions, data processing instructions, single data transfer instructions(LDR and STR). A data processing instruction, is encoded in the following manner: starting from left, the first 4 bits (31 to 28) are reserved for the condition field, Cond. The next two bits(27 and 26) are 00. The 25th bit is the immediate operand ,I. The next 4 bits (24 to 21) are kept for the Operation Code, OpCode. The 20th bit is for the Set condition,S. 19th to 16th bits are for the first operand register, Rn. The next four bits(15 to 12) are for destination register,Rd. The final end 12 bits(0 to 11) are for the second operand,Op 2.

For Branch instructions, the first 24 bits from right are for offset, followed by 4 bits then the last 4 bits i.e. 28 to 31 is the condition field.

For a single data transfer instruction, the order from right is as follows: Offset(0-11), Rd, Rn, L,W, B, U, P, I, 01, Cond. (L=Load/store bit, W=write-back bit, B=byte/word bit, U=up/down bit, P=pre/post indexing).

This concludes the 32 bit instruction formats.

The instructions given to us are in the Hexadecimal format.

If the ARM instruction is "0xEF000011" i.e. SWI 0x11, the simulator exits. It is shifted right by 26 bits, if the bit obtained at that position is 0, the instruction will follow the data processing (DP)format and processed likewise. If it is 1, it will follow a memory instruction (DT)format and Branch format if it is 2.

Decode will convert the Hexadecimal string to Binary. While following the ARM instruction set format it will find the corresponding fields in the instruction, and print the operation to be executed. The rest is upto the next stages of the pipelined architecture.

Suppose the instruction given is : e3a03002. The condition is e which translates to 1110, AL=always i.e. it will always execute irrespective of the other fields. The opcode is 1101 which is for MOV. The first operand is 0 which is R0 and destination register is R3. Immediate second operand is 2. Hence, the instruction after decoding is: MOV R3,R0

Execute

Initially, the general purpose registers have null values(i.e. 0). We plan to make a class for registers and make array of its objects for general purpose registers.

The instruction is only executed if the condition is true. After the command has been passed onto Execute function by giving the opcode and the various register addresses, we will have if else cases according to opcode for which sub functions like AND, ADD, etc. would be placed. The sub cases will in turn ask for register values for which the sub function will work. The method

then returns the value of the register or the memory to be written back. The execute function till then could hold the memory in temporary variables.

In branch instructions, since the branch has an address which we will scale down to indices in the array(memory in our analogy). We will jump the program counter over to that instruction's index and the instruction will get executed according to the Condition field. It will also store this address, if the branch has to go back to it in future. It also increments the program counter by 4.

After execute, the program will proceed to mem method.

Mem

This method is for the memory function, to decide if the instruction is for Load or Store. Depending on the OpCode it will Load a value from a register in the memory or store a value(the destination register) in the memory and display the corresponding message.

Write-back

Write back only concerns with writing the values in the desired memory and registers. Along with that this method will also print a message containing the result value of execute and the destination it has to be written to.

Testing

We will test the simulator by checking the output for multiple hex instructions. We can verify our results through hex to arm converters online.