Team Members

Deepak Srivatsav - 2016030
Yashit Maheshwary - 2016123
Anubhav Chaudhary - 2016013

Project Topic 1: ARM Simulator

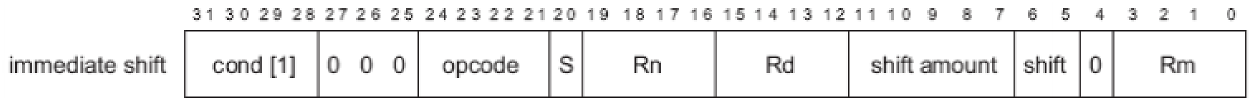
Understanding of the Problem:
This application must be able to analyse instructions and determine their operation, such as MOV, ADD, SUB etc. Furthermore it should be able to maintain a set of variables corresponding to registers, and should be able to trace and reflect the flow of execution in the program to produce the required result in the program. Jump and branch instructions may be implemented by use of the addresses provided in the file.

Methodology:

- An input file is fed to the program in which an assembly source code is coded in the format:
    - address<space>instruction
- A map may be used to assign various opcodes to a particular function. Similarly a set registers may be mapped in the same way to a set of variables.
- All the registers may be mapped to an array of integers which will always be displayed with every instruction.
- A small subset of instructions may also be displayed in which the current instruction will be highlighted.
- Jump instructions may be implemented by reading the address from the opcode and traversing through the file to find the appropriate address and execute the instruction.
- Every Instruction goes through the following stages:
  - Fetch: The instruction is fetched from the instruction memory
  - Decode: The instruction is decode to determine the operation to be performed
  - Execute: The arithmetic/ALU based operations are carried out in this stage
  - Memory: If any load/store operations exist, then the values are retrieved/stored from/to registers in this stage.
  - Write Back: If an ALU operation such as addition required writing back to a register, then that takes place at this stage

Plan

We will have an array for all the registers that are available with us. After every instruction the updated state of all the registers will be shown.
For a particular instruction we will find out to which set of instructions does it belong by checking the first four bytes of the instruction.
For eg. an immediate shift instruction look like this.



	31 30 29 28	27 26 25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8 7	6 5	4	3 2 1 0
immediate shift	cond [1]	0 0 0	opcode	S	Rn	Rd	shift amount	shift	0	Rm

Having a separate set of bits which will include CS(Carry Set), EQ(Equal or Zero Set), VS(Overflow Set), etc. to handle the ARM instructions which will be execute conditionally. As for the various changes that will occur in the registers we will simply update the value of the register in the array in which we are storing the registers.

Extra features

Implementation of a GUI for the same, similar to that of ARMsim. We will also try to include display of various flags at different stages of execution. We will also try to implement breakpoint features.

Timeline

30th October: completion of implementation of basic add, sub, mov functions.
10th November: completion of implementation of conditional statements
16th November: completion of implementation of jump statements
25th November: completion of additional features such as GUI.

[POST MID-REPORT]
The following instructions are implement in the final project.

1. MOV - Move one register's value into another
2. Add - Add values of registers/immediate values.
3. SUB - Subtract values of registers/imm
4. RSB - Reverse Subtract.
5. EOR - XOR two values in registers.
6. ORR - OR two values in registers.
7. AND - AND two values in registers.
8. BIC - performs an AND operation on the bits in Rn with the complements of the corresponding bits in the value of Operand2
9. MVN - moves the complement value of one register into another.
10. CMP - compare's the values in two registers and sets the flags accordingly.
11. Branch Instruction - Followed by a compare instruction,
12. SWI Read/Write Integer, String - Reads and writes from/to console.
13. SWI malloc allocation - Requests memory from kernel
14. Load/Store - Load store values stored in the memory.

A map is used to assign various opcodes to a particular function. Similarly a set registers is mapped in the same way to a set of variables.
All the registers are mapped to an array of integers which will always be displayed with every instruction.
Jump instructions are implemented by reading the address from the opcode and traversing through the file to find the appropriate address and execute the instruction.
Every Instruction goes through the following stages:

- Fetch: The instruction is fetched from the instruction memory
- Decode: The instruction is decode to determine the operation to be performed
- Execute: The arithmetic/ALU based operations are carried out in this stage
- Memory: If any load/store operations exist, then the values are retrieved/stored from/to registers in this stage.
- Write Back: If an ALU operation such as addition required writing back to a register, then that takes place at this stage

So first the instructions are fetched from the .MEM file and are decoded as per the ISA rules. After decode is done the instruction is passed to a function which decodes the operands and immediate values (if any). Then that instruction is executed various flags are set, registers updated.
For branch instruction both forward and backward branching is working. SWI functions include memory allocation from malloc().

We also have a GUI which shows all the flags and register values along with the instructions and what our program is outputting. It also shows the current flags.