CO Final Project Report

Team Members:
Mehak Gupta (2016163)
Simran Deol (2016097)
Zoha Hamid (2016124)

ARM-De-Simulator

November 29th, 2017

Contents:

Chosen Project topic:

Arm Simulator (Project 1)

Design and implement the function simulator in Java for subset of ARM instructions

Project Methodology:

1. Making a function to convert the hex instruction format to binary.
2. Store the required opcodes in a hashmap.
3. Extract the required bits and find the opcode
4. Make a separate hashmap for the function codes

5. Current values of the registers would be stored in an array.
6. We will use conditional statements to decide which operation will be performed and when.
7. These operations will change the value of the registers in the register array.
8. And till we come across the exit code the program will continue to run.

Initial Project Plan:

First we will begin with reading the instruction set of Arm Assembly. List the opcodes and function codes. Find the different instruction formats. Then we will code the functions. Test the arm simulator. Then make documentation.

Final implemented solution:

We have made an ARM-De-Simulator, which is a functional subset of ARM Processor. It supports the set of ARM commands done in class.

ARM-De-Simulator Design and Documentation

Cond	F	I	Opcode	S	Rn	Rd	Operand2
31-28	27-26	25	24-21	20	19-16	15-12	11-0
4 bits	2 bits	1 bit	4 bits	1 bit	4 bits	4 bits	12 bits

Data Processing:

Cond	**F**	**Opcode**	**Rn**	**Rd**	**Offset**
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits

Data Processing Instructions Implemented:

1. ADD Rd, Rn, Rm | ADD Rd, Rn, #imm
2. SUB Rd, Rn, Rm | SUB Rd, Rn, #imm
3. AND Rd, Rn, Rm | AND Rd, Rn, #imm
4. ORR Rd, Rn, Rm | ORR Rd, Rn, #imm
5. CMP Rn, Rm | CMP Rn, #imm
6. MOV Rd, Rm | MOV Rd, #imm
7. MNV Rd, Rm | MNV RD, #imm

Data Transfer:

Data transfer instructions implemented:

1. LDR Rd [Rn, #imm]
2. STR Rd [Rn, #imm]

Cond	**F**	**Opcode**	**Offset**
4 bits	2 bits	2 bits	24 bits

Branch:
Branch Instructions Implemented:
BEQ - Branch if equals
BNE - Branch if not equal
BLT - Branch if less than’BLE - Branch if less than equal
BGT - Branch if greater than
BGE - Branch if greater than equal
BAL - Branch and link

Instruction Mnemonics and Summary:

ADD	Add	Rd := Rn + Op2
AND	AND	Rd := Rn AND Op2
SUB	Subtract	Rd := Rn - Op2
BAL	Branch and Link	R14 := R15, R15 := address
CMP	Compare	CPSR flags := Rn - Op2
ORR	OR	Rd := Rn OR Op2
MOV	Move register or constant	Rd : = Op2
MVN	Move negative	Rd := 0xFFFFFFFF

	register	EOR Op2
LDR	Load register from memory	Rd := (address)
STR	Store register to memory	<address>:= Rd
MUL	Multiply	Rd := Rm * Rs
SWI	Software Interrupt	OS call

Simulator workflow:
1. Memory is loaded with input memory file.
2. Simulator executes instruction one by one.

The simulator exits only when the instruction sequence reads “SWI 0x11”. The implementation of fetch, decode, execute, memory, and write-back function are implemented as separate functions in Java.

FETCH: Reads from the instruction memory and updates the instruction register
DECODE: Reads the instruction register, reads operand1, operand2 from the register file, decides the operation to be performed in execute stage
EXECUTE: Executes the ALU operation based on ALUop MEM: Performs the memory function
WRITE-BACK: Writes the results back to register file.

<u>Fetch</u>:
1. The instruction is fetched from the .mem file which is the input at the time of execution. The .mem file would contain the address as well as the instruction (in hexadecimal format) of the ARM assembly command.
2. This instruction is written into the MEM array.

<u>Decode</u>:
1. This function uses bit-masking to calculate the flag, the opcode, the condition bits and the immediate of the given instruction or one that is fed into the code.
2. We know that if immediate=0, the second operand is a register, and if immediate =1, the second operand is a constant value. Using this, and the opcode, we determine the operation in the instruction as well as operands 1 and 2.

Execute:
1. This function executes the instruction decoded, with the instruction’s operands and the destination register.
2. If the instruction is ADD, this function computes operand1 plus operand2 and places it in a temporary variable.
3. For instructions like LDR, STR, there are different arrays for each register, in order to store or load values from memory (registers) we place values from respective registers and put them in respective registers.
4. For instructions like BRANCH, we increment the PC or R[15], by using the formula R[15]/PC = R[15]/PC + 1 to compute the correct branch address.

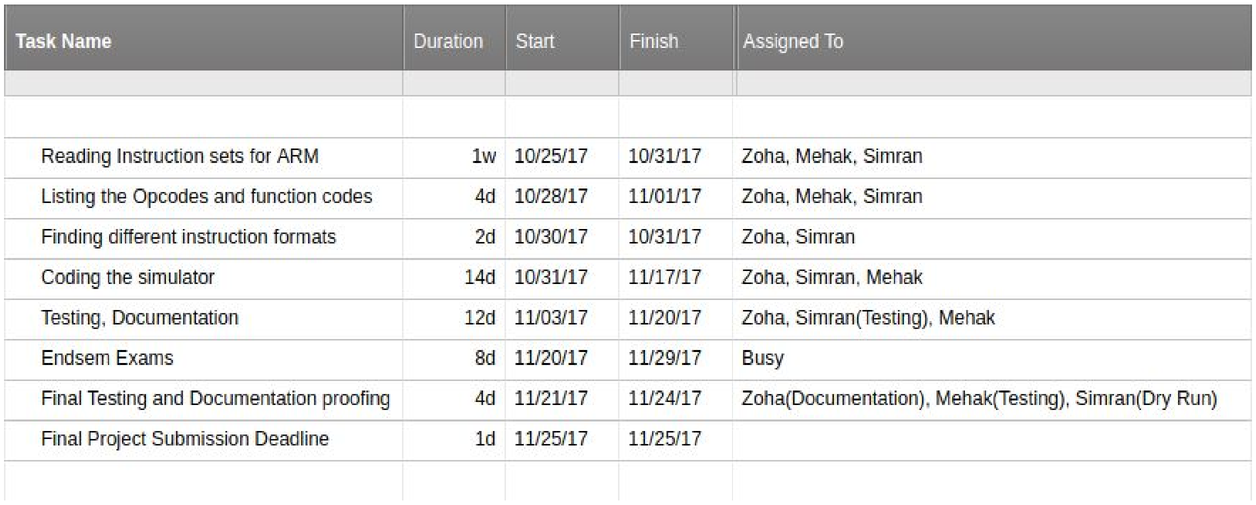
Memory:
1. This function elaborates what is to be written to memory, or loaded from memory.
2. In all instructions except LDR, STR, there's no memory function being carried out. Thus, it only prints statements for LDR and STR printing stored values and destinations.
Writeback:
1. This function writes back to the destination register, what is stored in the temporary variable.

2. If there is no writeback for example in case of Branch and CMP instruction, it prints that “No writeback operation is required”.

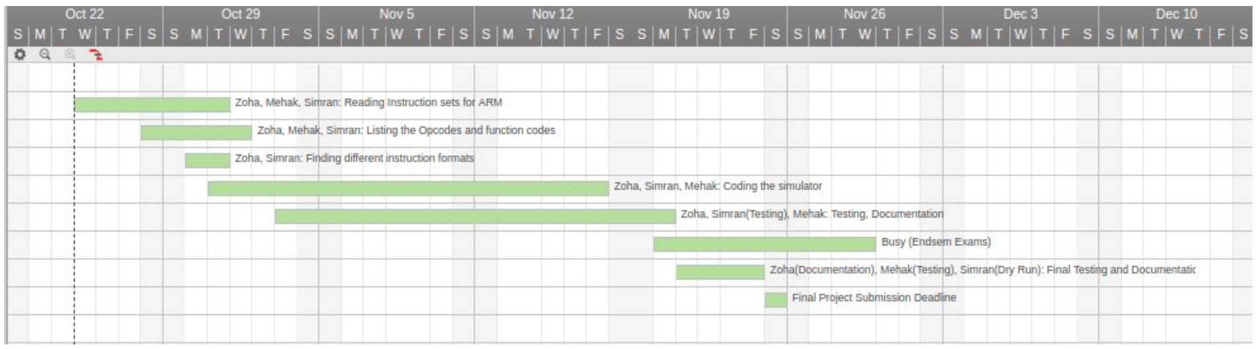
===

Project Timeline:

The project was done according to the timeline:



Task Name	Duration	Start	Finish	Assigned To
Reading Instruction sets for ARM	1w	10/25/17	10/31/17	Zoha, Mehak, Simran
Listing the Opcodes and function codes	4d	10/28/17	11/01/17	Zoha, Mehak, Simran
Finding different instruction formats	2d	10/30/17	10/31/17	Zoha, Simran
Coding the simulator	14d	10/31/17	11/17/17	Zoha, Simran, Mehak
Testing, Documentation	12d	11/03/17	11/20/17	Zoha, Simran(Testing), Mehak
Endsem Exams	8d	11/20/17	11/29/17	Busy
Final Testing and Documentation proofing	4d	11/21/17	11/24/17	Zoha(Documentation), Mehak(Testing), Simran(Dry Run)
Final Project Submission Deadline	1d	11/25/17	11/25/17	



Individual Contributions:

Mehak: Reading Instruction sets for ARM + Listing the Opcodes and the functions + Finding the different instruction formats +Coding the simulator + Testing + Final testing and implementation
Simran: Reading Instruction sets for ARM + Listing the Opcodes and the functions + Finding the different instruction formats + Coding the simulator + Final testing and Dry run
Zoha: Reading Instruction sets for ARM + Listing the Opcodes and the functions + Coding the simulator + Documentation + Final testing
Bonus attempted: We have also implemented the MUL function which has a completely different instruction format.