ARM SIMULATOR

Objective:
To design and implement ARM the function simulator in JAVA for a subset of ARM instructions.

Implementation:

1. Reading instructions from .MEM file:
   Using **ChooseFIle** class(GUI-JavaFX) we can select the file we need to read. Then that file is read using BufferedReader which reads the file line by line and stores it into a string array.

2. Storing and Converting the hex code in binary:
   Hex code was read as a String and was stored in an array, the first three characters of the hex code will tell us about the address of the instruction and the remaining code can be converted to binary to find the type of instruction. A separate **Instruction class** is made to store each instruction as an individual object, and a dictionary(HashMap) is generated having all the instructions in the given file.

3. Finding opcode, destination and source registers form the instruction:
   The opcode field from bits 24-21 tells us exactly which type of instruction. To find the type of instruction we will need to maintain a list of opcodes for the instructions and then compare the values in this list with that of the given instruction. Bits 19-16 tell us about the first source register. Bits 15-12 tell us destination register and bits 11-0 tell us about the second operand which can either be a register or it can be an immediate value. All these values are stored in the Instruction(object) as its instance variables and is used for further decoding.

4. Defining Functions:

- **Fetch**: Fetch function gives the next instruction from memory with the help of program counter or the branch command, which may point to the current instruction. Hex code and its address of the instruction in memory is printed.

- **Decode**: After fetching we decode the instruction, identification of the instruction is done by matching the instruction's opcode. Then according to the instruction type we have further conditions for identifying operands, if there are any.

- **Execute**: For the decoded instruction, we get the operands of that instruction and then perform the appropriate function based on the Opcode of the instruction is performed and the final result is stored in the destination register.

- **Memory**: This function tells us whether there is a memory operation being done by the instruction executed or not which only happens when load or store instruction is called.

- **WriteBack**: After instruction, this method tells us the which value is written in the register.

5. Printing the output for each stage:
   For any instruction we have to output the work done by a given instruction across the five stages of the datapath namely- FETCH, DECODE, EXECUTE, MEMORY, WRITEBACK. For example for the given instruction- 0x0 0xE3A0200A, the output will be as follows-

   ```
   FETCH instruction 0xE3A0200A from address 0x0
   DECODE: Operation is MOV, First Operand is R0 and immediate second operand is 10.
   Destination register is R2.
   EXECUTE: MOV 10 in R2.
   MEMORY: No memory operation.
   WRITEBACK: Write 10 to R2.
   ```

After we will have to store the value in R2 in a variable as it might be required in some instruction later on.

Problems Faced:

- Finding opcodes of the instructions: Since the lecture slides and notes only mentioned some of the opcodes of instructions, finding the opcodes of some other instructions was one of the problems faced.
- Implementation of branching, determining the address of label was quite difficult, maintaining program counter was quite hectic.

Bonus:

- Graphical User Interface is added in the Project to browse the .MEM file (in which the hex code is written) from the computer, hence the path of the file need not to be updated every time to simulate other .MEM files.
- Branching is also implemented, hence simple loops can also be run on the ARM Simulator.
- Object Oriented Programming is used in the implementation of ARM-Simulator.

Kshitiz Jain (2016051)
Nikhil Sachdeva (2016061)
Apoorv Khattar (2016016)