# CO Project

## Documentation

Made By:

Mansi Goel – 2014062

Taneea S Agrawaal - 2014166

The ARMSim has been designed in the following manner:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Cond** | **F** | **I** | **Opcode** | **S** | **Rn** | **Rd** | **Operand2** |
| 4 bits | 2 bits | 1 bit | 4 bits | 1 bit | 4 bits | 4 bits | 12 bits |

**Data Processing Instructions Implemented:**

* ADD Rd, Rn, Rm | ADD Rd, Rn, #imm
* SUB Rd, Rn, Rm | SUB Rd, Rn, #imm
* ADD Rd, Rn, Rm | ADD Rd, Rn, #imm
* SUB Rd, Rn, Rm | SUB Rd, Rn, #imm
* AND Rd, Rn, Rm | AND Rd, Rn, #imm
* ORR Rd, Rn, Rm | ORR Rd, Rn, #imm
* ADC Rd, Rn, Rm | ORR Rd, Rn, #imm
* CMP Rn, Rm | CMP Rn, #imm
* MOV Rd, Rm | MOV Rd, #imm
* MNV Rd, Rm | MNV RD, #imm

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Cond** | **F** | **Opcode** | **Rn** | **Rd** | **Offset12** |
| 4 bits | 2 bits | 6 bits | 4 bits | 4 bits | 12 bits |

**Data Transfer Instructions Implemented:**

* LDR Rd [Rn, #imm]
* STR Rd [Rn, #imm]

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

**Branch Instructions Implemented:**

* BEQ
* BNE
* BLT
* BLE
* BGT
* BGE
* BAL

**FETCH:**

1. The instruction is fetched from the .mem file which is inputted 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.

**DECODE**:

* 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 called result.
  3. For instructions like LDR, STR, there are different arrays for each register, in order to store or load values from memory(which are registers, in this case) we take/place values from respective registers and put them in respective registers.
  4. For instructions like BRANCH, we sign extend the 24 bits of the offset to 32 bit, and then increment the PC or R[15], by using the formula R[15] = R[15] + 8 + SignExt32(Offset\*4) 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 out the stored/loaded values and their destinations.

**WRITEBACK**:

* 1. This function writes back to the destination register, what is stored in the temporary variable ("result").
  2. If there is no writeback for example in case of branch and cmp instruction, it prints that “No writeback operation is required”.