**GROUP\_1\_README**

**PHASE 2**

**How to execute Phase 2:**

* .mc file generated by phase1 must be in the same folder.
* Compile in terminal using command “g++ phase2.cpp”.
* Execute using “.\a.out” in Ubuntu and “a” in windows { after compiling “phase2.cpp” }.
* “ data\_out.mem ” created as output.

This phase consisted of **fetch** stage,**decode** stage, **execute** stage, **memory access** stage and **writeback** stage.

**Work Split:**

**1. Prateek Mehandiratta (2017csb1098):** Fetch & Decode Stage.

* All the instructions and data is stored in the memory. (Instruction memory-> 0 to 1999, Data memory-> 2000 to 2999, Stack memory-> 3000 to 3999)
* Initially, PC is set to address 0x0 and in fetch stage , IR is reading the content at PC and PC is incremented by 4.
* In decode stage, Instructions are classified in their respective formats (R,I,S,SB,U,UJ) by decoding the opcode of inst. and then the different fields are extracted according to the format.
* Registry file is read thereafter.

**2. Vaibhav Singh (2017csb1117):** Execute Stage.

* Assigned control line numbers to each instruction.
* Then using decode stage, obtained the control line number and executed the instruction accordingly.
* Wrote separate code chunks for each instruction.
* Updated the required values in intermediate register Rz after calculating it so that it can be used in memory access stage.

**3. Abhinav Khanna (2017csb1061):** Memory Access & Write Back Stage.

* Used the values stored in register Rz and Rm to store or load into memory array.
* Depending on the Control Signals generated, different load and store instructions were performed
* For Write back stage, again Control Signals were used to determine whether writeback was required, and if so, the number of the register.
* Rz: Stored memory address to be executed
* Rm: Element to be stored in the memory
* Ry: Number of the register to be written

**Instructions not supported:**

* Pseudo instructions
* fence, fence.i
* ecall, ebreak
* CSRRW, CSRRC, CSRRS, CSRRWI, CSRRSI, CSRRCI