++RISC V Simulator

~ By Team “HAKUNA MATATA”

Phase 1

- In this Phase we took different types of assembly codes as Input files and we Assumed Registers as Integers and memory as a string array of size 4096.

- Divided the memory of 4096 elements into Two equal Parts one for each core i.e from 0 to 2047 core1 and 2048 to 4096 core2.

- The memory allocated for each core is also divided into two parts, one for storing all instructions and other for storing data segments.

- So for core1 we took 0 to 1000 places in memory for storing instructions and 1000 to 2047 for storing data segments and for core 2 2048 to 3047 places in memory for storing instructions and 3048 to 4096 for storing data segments.

- So in the processor class we declared memory and inherited classes like :

    1. load_program → to load all instructions into memory
    2. load_data → to load all data segments like array strings into memory
    3. Run → to execute the loaded instructions parallelly from core1 and core2

- In the core we declared 32 registers and inherited classes like :

    1. labels → used a hashmap to store all the labels and their address
    2.execute → to execute all the instructions stored in memory

- Finally printing the memory and sorted output in the memory.txt and output.txt respectively.

Phase 2

- In this phase we tried to implement a pipeline for the simulator which we have done in phase 1.

- For that we have considered new classes like:
    - pipeline control → which is used to incorporate pipelining with 5 stages (IF,ID,EXE,MEM,WB) with the added functionality to enable and disable data forwarding and in stages the instruction is passed from the previous register.
    - We also used Buffer Class to store the data that was extracted from each stage.

Instruction Fetch (IF):

- In this stage, the processor fetches the instruction from the memory using PC.The fetched instruction is then passed to the next stage. The PC is incremented to point to the next instruction.

Instruction Decode (ID)

- The fetched instruction is decoded in this stage. The processor determines the type of instruction, identifies the operands, and prepares for the execution stage. Register values are read from the register file during this stage.

Execution (EX) Stage:

- The actual computation specified by the instruction takes place in this stage. For arithmetic and logic instructions, this is where the calculations occur. For control flow instructions, the branch conditions are evaluated here.

Memory Access (MEM) Stage:

- If the instruction involves a memory operation, such as a load or store, it is performed in this stage. Data is read from or written to the memory..

Write Back (WB) Stage:

- The final results of the instruction are written back to the register file in this stage. Both with forwarding and without forwarding have been implemented.

Stalls and IPC

- We were able to calculate the Stalls and IPC values of each core separately by calculating the No of Instructions and No of cycles taken.

- We mainly found stalls due to Hazards like data dependencies and miss predictions.

Branch and Miss Prediction

- In this phase we took prediction as always not taken and if it wrongly predicted we removed the present instruction and fetched the new instruction based on PC value.

Latencies

- We tried to implement latencies in this phase but ended up in getting errors and wrong no of stalls..

Phase 3

- In this phase we have tried to fix the problems faced in phase 2 and tried to implement the cache to the simulator.
- We incorporated the phase 2 with two different cache replacement policies one is LRU and other is MRU.
- Created a class named cache, containing input along with the cache size, block size, associativity and access latency of the cache.
- By using that class we are able to divide the address into tags ,number of bits and index.
- By using access boolean we are able to know that the address is already in the cache or not.
- We were able to calculate the hit rate but with the miss rate we ended up with wrong answers because of errors in phase 2 such as incorrect branch predictions..etc

Phase 4

- In this phase we both sat together and tried to find out the problems present in the previous phases and started to fix them.

  The problems we came through and tried to solve them are
  - Implementation of multi level cache.
  - Making predictions more accurately.
  - We got less IPC than the original IPC, but compared to our previous phases we improved it.

- Making effective data forwarding of instructions during Hazards.

❖ But with the multilevel cache implementation we wrote a pseudo code, which when we tried to merge with the original one, we ended up with errors. So we left it in another file with the code we tried.