MIPS32 Simulator Proposal

> **Name:** Ekansh Somani | **Roll:** 20110065
>
> **Course:** CS 215 - Computer Organization and Architecture
>
> **GitHub Link:** MIPS32-Simulator | **Team Name:** Lone-wolf

My Understanding of the problem statement

Design a simulator for 32 bits MIPS/ARM processor (I choose MIPS). It must take in all the instructions from a file containing a sequence of 32 bits instruction codes, and then execute them. It also needs to have memory mapped IO and the output needs to be displayed on the screen.

MIPS32 Processor and ISA Basic Details

- "MIPS architecture is based on a fixed length, regularly encoded instruction set. ... [where] ... All the operations are performed on operands in processor registers, and main memory is accessed only by load and store instructions." $[1.a]$
- **ISA Releases**: MIPS32 has had 6 ISA releases, with major revisions in Release-2 and Release-6.$[1.a]$
- **Instruction Categories**: Instructions are divided into Load and Store, Computational, Jump and Branch, and Address Computation and Large Constant Instructions. Miscellaneous instructions include Exception, Conditional Move, Prefetch, NOP, and Serialization.$[1.b]$
- **Registers**: 32 general-purpose registers, a program counter (`PC`), cycle count (`$count`), and HI and LO registers.
- **Coprocessors:** support upto 4 coprocessors named as:

  - `cp0`: System Control Coprocessor for control, memory management, and exception handling.
  - `cp1`: Floating point Coprocessor.
  - `cp2` and `cp3` (Custom Implementation)
- **Delayed Branches:** Original MIPS architecture required delayed branches, $[1.c]$ resolved in Release-6.
- MIPS stands for **M**icroprocessing without **I**nterlocked **P**ipeline **S**tages.

Simulator Design (Tools and Technology)

- **ISA Choice:** Using Release-6 ISA, supporting all Instructions.
- **Coprocessors:** Including `cp0` (essential) for exception handling and memory management. `cp1`, `cp2` , `cp3` not included.
- **Memory**: 8 MiB segment (1024*1024 containers of 32 bits).
- **Pipeline Stage:** Four stages - fetch, decode, execute, write back. Register memory operations occur during execution.
- **Clock Concept:** Operates on a cycle-by-cycle basis, where each cycle represents a clock tick. Initially, an Instruction completes one stage in a cycle irrespective of the time complexity. (pipelining gives us 1 CPI)
- **Memory Mapped IO:** Two options:
  - import all the input from a file onto memory at the start and take input from there. Writes output onto memory, that we export into another file at the end.

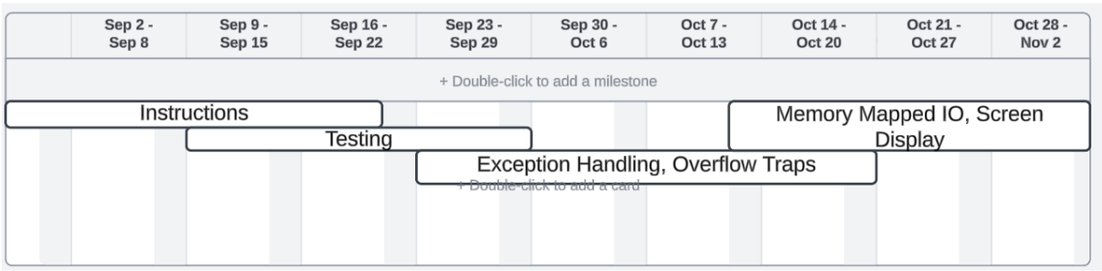
    - Run a parallel program for real time I/O via memory mapping between programs.
- **Language:** C++20 for simulator. Maybe Python 3.12.2 for generating binary instructions and memory mapped IO.
- **Code Editor:** VS Code. **OS:** Windows 11 (ubuntu with wsl-2 if needed). **Version Control:** Git 2.44.0.windows.1. **Compiler:** g++13.2.0.

Additional Features (if time permits)

- **Clock:** Implement variable cycles for different operations, introducing a more realistic processor simulation.
- **Cache:** Implement a small 4-8 KiB cache to understand its functionality.

Key Milestones/Task List

- Coprocessors, Registers, Memory objects created. A basic process of fetch, decode, and execute ready. (31 Aug)
- Instructions Complete without pipelining, overflow traps and exception handling. (20 Sep)
- Testing Complete: Prepare a few binary programs either directly or by using an assembler (without exceptions). And test the code to see how it performs. (29 Sep)
- Pipelining, Exception Handling, Overflow Traps Complete. (19 Oct)
- Memory Mapped IO, Screen Display systems. (2 Nov)



Learning Outcome

- I didn’t know much about processors at the start of this project, and I have already learned a lot. I have learned how the RAM is managed for a program, how instructions are fetched and executed. How registers store data, and the many ways instructions can interpret it. I know what kinds of instructions are presented at the assembly and binary level now.
- I expect to have better understanding of how a processor functions, how interrupts, and handling of exception work, and what do coprocessors handle by the end of this project. I also expect to learn more about Memory Mapped IO, and learn and implement exactly how heaps and stacks work in memory.

Reference

1. MIPS® Architecture For Programmers Volume I-A: Introduction to the MIPS32® Architecture Revision 6.01
    a. Chapter 2: Overview Of the MIPS Architecture. 2.1 Historical Perspective [Pg 21]
    b. Chapter 5: CPU Instruction Set.
    c. Chapter 5: CPU Instruction Set. 5.3 Jump and Branch Instructions [Pg 61]