Project 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." $[1.a]$
- The MIPS32 Architecture has had 6 releases of ISAs' of which Release-2 and Release-6 had major revisions.$[1.a]$
- The instructions for main MIPS32 processor can be divided into four major categories: Load and Store, Computational, Jump and Branch, and Address computation and large constant Instructions. There are some miscellaneous instructions which don't fall into any of the above categories. Miscellaneous instruction include: Exception, Conditional Move, Prefetch, NOP and Serialization Instructions.$[1.b]$
- MIPS32 Processor has 32 general purpose registers, a `PC` (program counter), `$count` (cycle count) and `HI` and `LO` registers.
- MIPS32 ISA also support upto 4 coprocessors named as:

  - `cp0`: System Control Coprocessor. Provides control, memory management, and exception handling functions.
  -
  `cp1`: Floating point Coprocessor. Load and store instructions (move to and from coprocessor), reserved in the main opcode space. Coprocessor-specific operations, defined entirely by the coprocessor.
  -
  `cp2` and `cp3` (Custom Implementation)
- The original MIPS architecture not just supports but rather requires delayed branches.$[1.c]$ This creates an ambiguity for our simulator because delay slot would depend on the number of stages we have in the pipeline. This (thankfully) is resolved by MIPS32 Release-6 ISA.
- MIPS stands for Microprocessor without Interlocked Pipeline Stages.

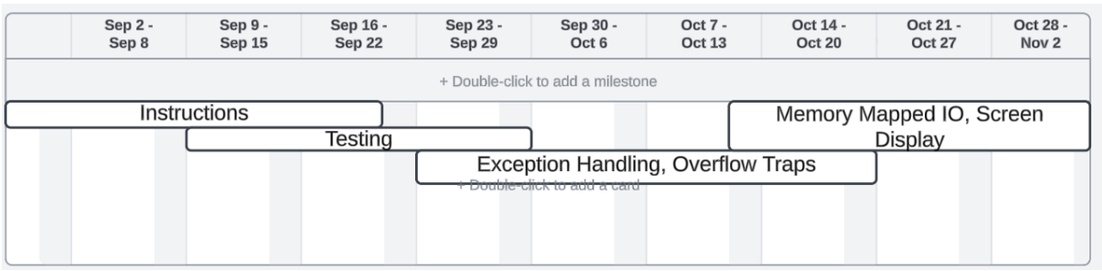
For my simulator, (**Tools and Technology**)

- I have decided to use Release-6 ISA for the processor.
- I will include `cp0` System Control Coprocessor because of the ease of exception handling, memory management and kernel control. I would attempt to also include `cp1` Floating Point Coprocessor if I have time at the end.
- Pipeline would only have one stage that would perform fetch, decode, execute, memory and write back.

- My memory segment would be of 8 MiB size. (1024*1024 containers of 32 bits width).
- I am slightly confused about memory mapped IO and have come up with two options:
  - import all the input (that would be required while the simulator would run) from a file onto memory and take input from there. It writes output onto memory, that we export into another file at the end.
  - Run a separate program, that's constantly running in parallel with the simulator, that takes input from keyboard and immediately loads it onto a section of memory that our simulator can read. And there is another section of memory where our simulator can write the output, and the program will be constantly reading that memory and displaying that output on the screen.
- Language: C++20 for simulator. Maybe Python 3.9 for generating binary instructions and memory mapped IO.

Key Milestones/Task List

- Coprocessors, Registers, Memory objects created. A basic process of fetch, decode, execute, memory, and write back (one stage pipeline) ready. (31 Aug)
- Instructions Complete without 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)
- Exception Handling, Overflow Traps Complete. (19 Oct)
- Memory Mapped IO, Screen Display systems. (2 Nov)



Work Done Until now (git)

First Milestone reached. ALU Release-2 ISA version Complete. I had originally planned to create for release-2 instructions until I encountered problems while starting to write Jump and Branch instructions. So I am yet to convert the ALU for Release-6. The will likely not just remain an ALU but rather include all computational instructions when I convert it. I have written functionPointers.py python file in a bid to figure out how exactly to decode instructions. It will then be translated into C++ header file (.hpp extension).

Note: I have only included basic details about MIPS processor and ISA here. Since, I felt like it wasn't purpose of this proposal to go into those details, but that doesn't imply that I have not read those details. The README.md file on the GitHub repository shows all the research that I have done for the project till now.

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]