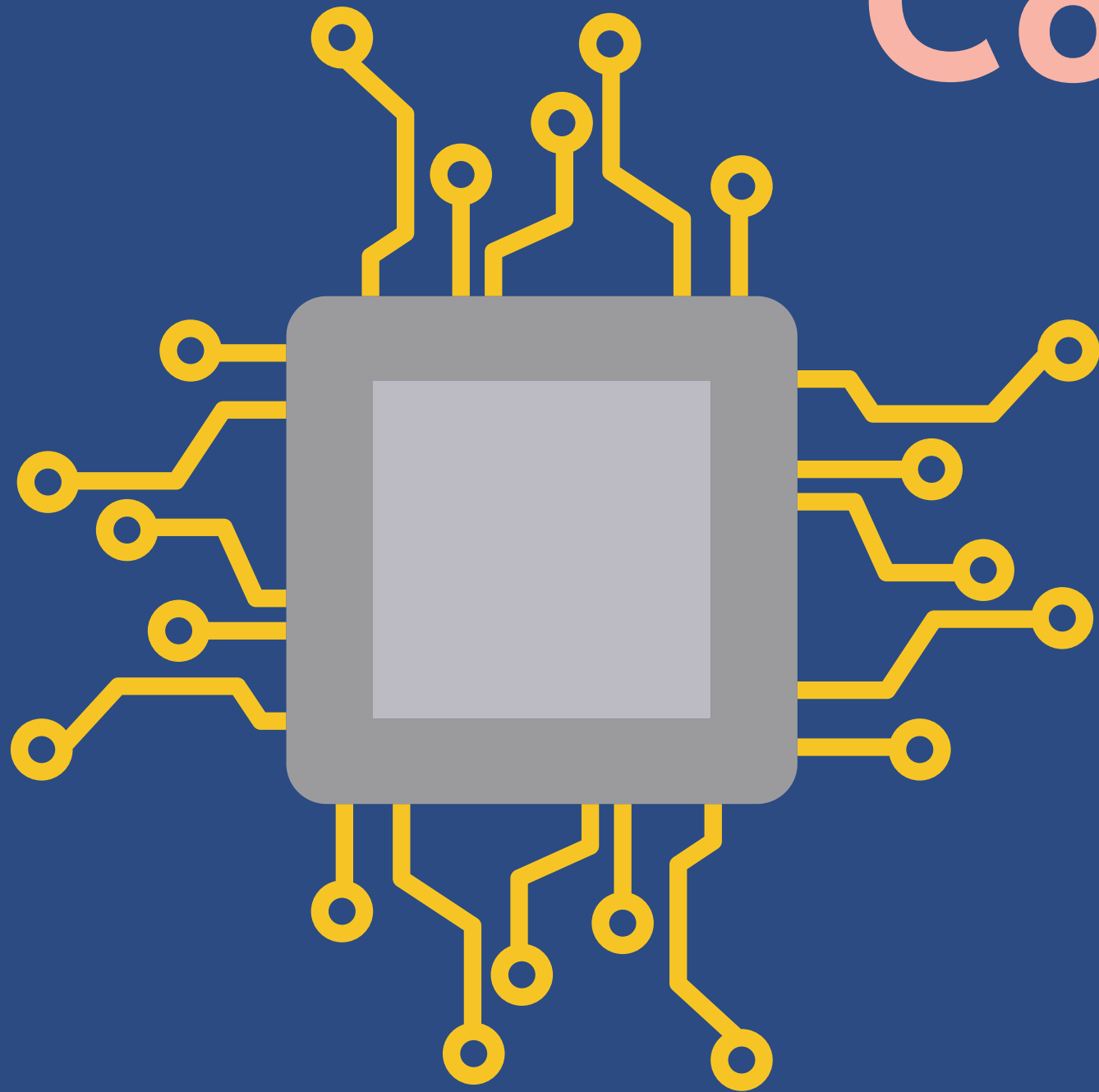
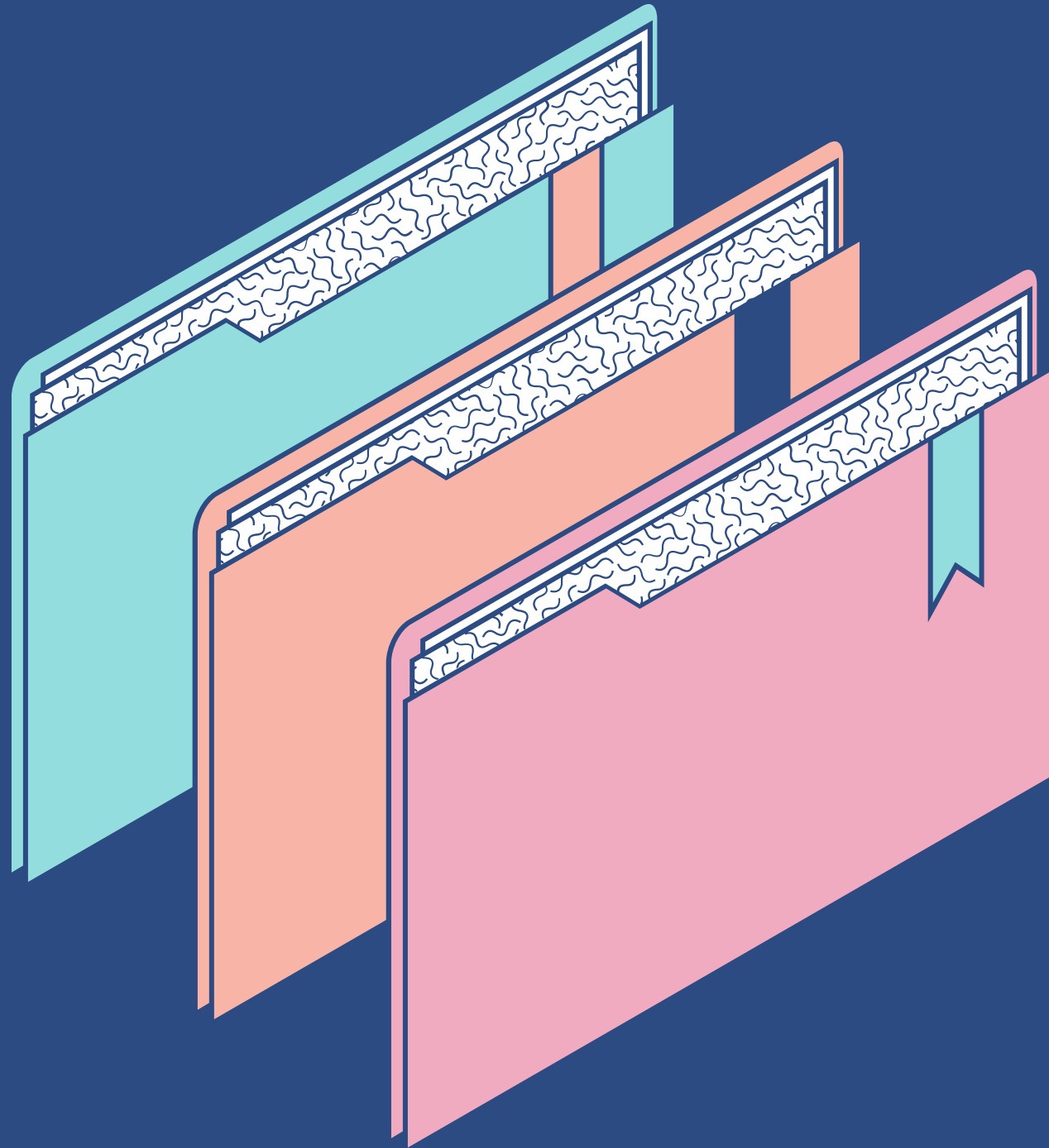




RISC-V CPU Core in VHDL



Content



- WHAT IS RISC-V?
- APPLICATIONS
- PROFILES AND PLATFORMS
- PROJECT OVERVIEW
- COMPONENTS
 - ADDER, ALU, DECODER, MUX
 - DATA & INSTRUCTION MEMORY
 - REGISTER FILE
 - SIGN EXTENDER
- CONCLUSION



What is RISC-V?

RISC

Reduced Instruction Set Computer

RISC-V is an open standard instruction set architecture (ISA) based on established (RISC) principles.



- Simple Instructions and decoding
- Instruction comes undersize of one word
- Instruction takes a single clock cycle for execution
- More General Purpose Registers
- Simple Addressing Modes
- Fewer Data Types
- A pipeline can be achieved



Open Source

RISC-V is unique, free, open-source ISA to which software can be ported, hardware can be developed, and processors can be built to support it. It is driven through open collaboration, which enables freedom of design across all domains and industries.

Speaking of industries....

Applications



The Application of RISC-V processor include following:

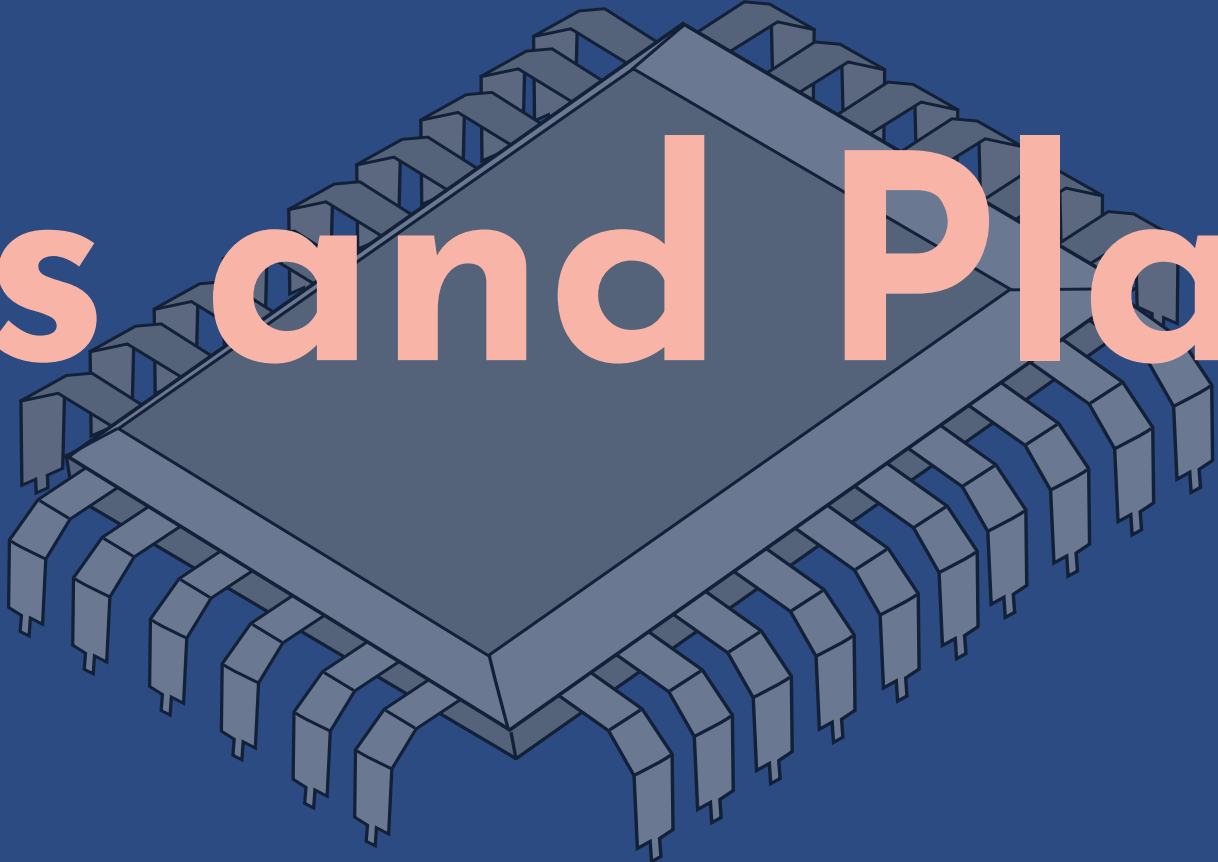
RISC-V is mainly used in Embedded Systems, Artificial Intelligence and Machine Learning.



- High-Performance-Based Embedded Systems
- Edge Computing
- Artificial Intelligence & IoT
- Storage Applications
- Cloud Servers

All the way from small microcontrollers to **Desktop PCs, Supercomputers and Vector Processors**

Profiles and Platforms



ISA Profiles

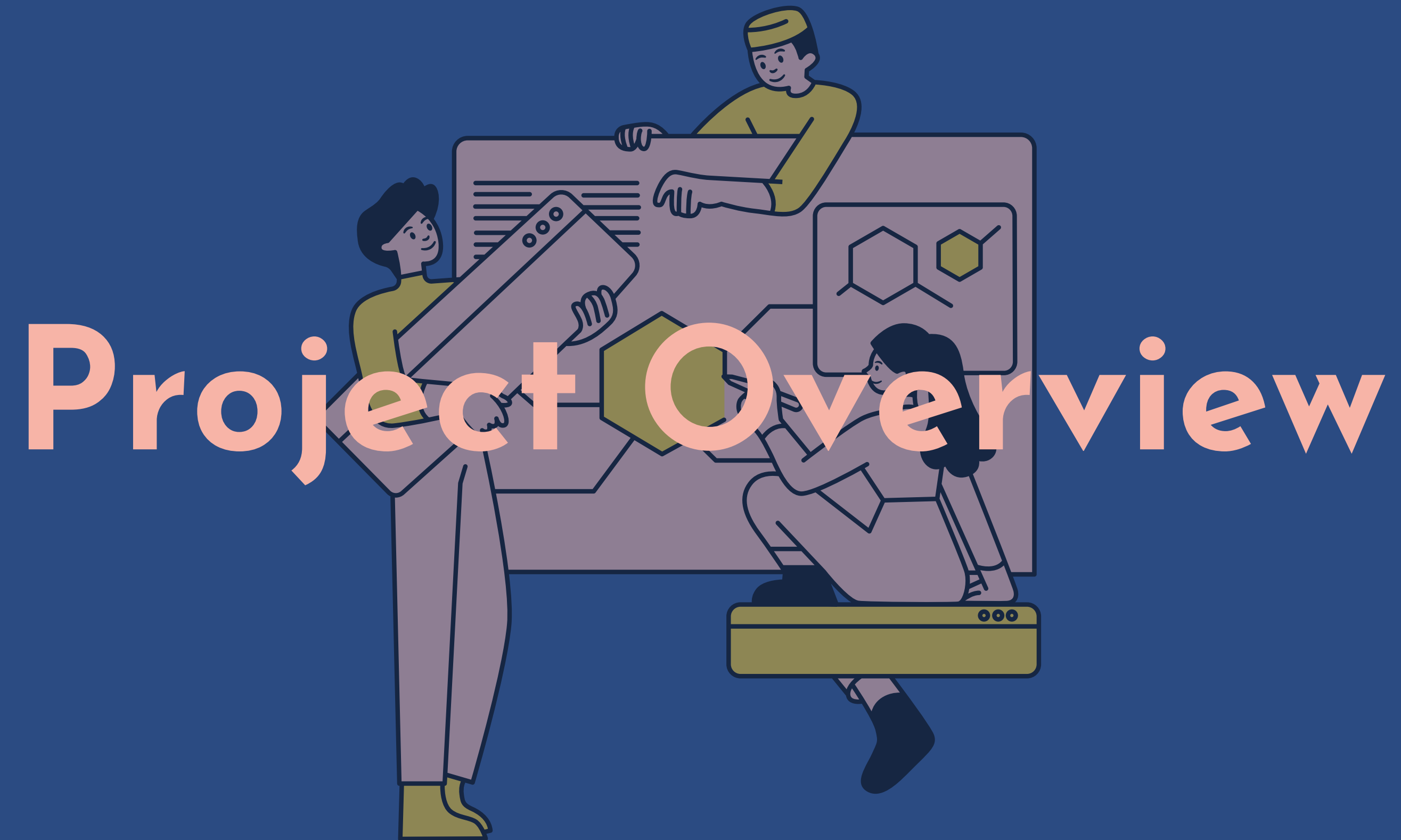


- A set of compatible **extensions**
- **Extension** types: required, optional, unsupported, incompatible
- **Profile** types:
 - Application (RVApp): Linux-class and other embedded designs with complex ISA needs
 - Microcontroller (RVMC): Cost-sensitive application-optimized embedded designs running bare-metal or RTOS environments
- Running the same sequence of instructions between implementations are RISC-V compatible.

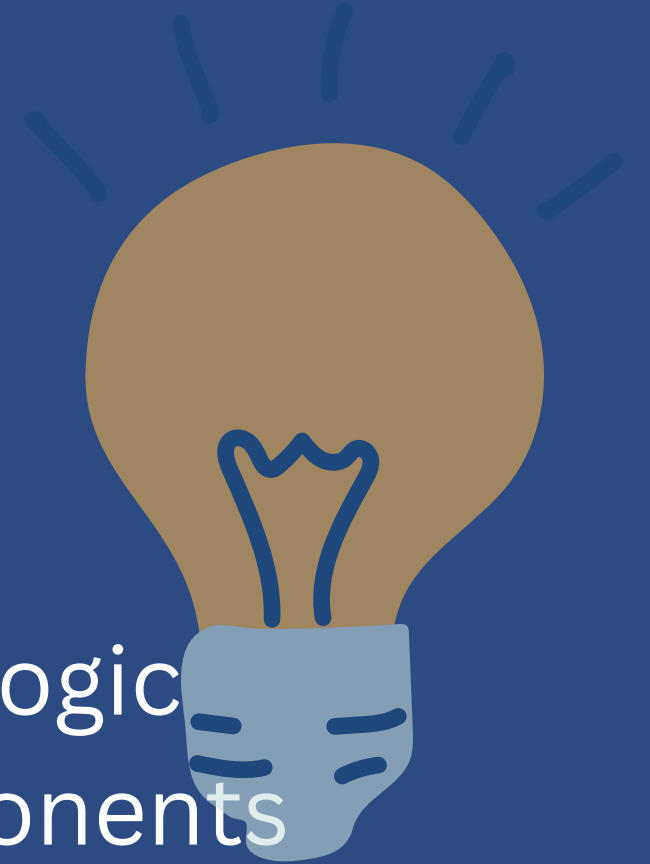
System Platforms



- A set of compatible features
- **ISA Profiles**, SW and HW system components, standardized HW/SW interfaces.
- **Platform** types:
 - OS / A
 - M
- Ability to move an executable from one implementation to another and get the same results are RISC-V compatible



Introduction

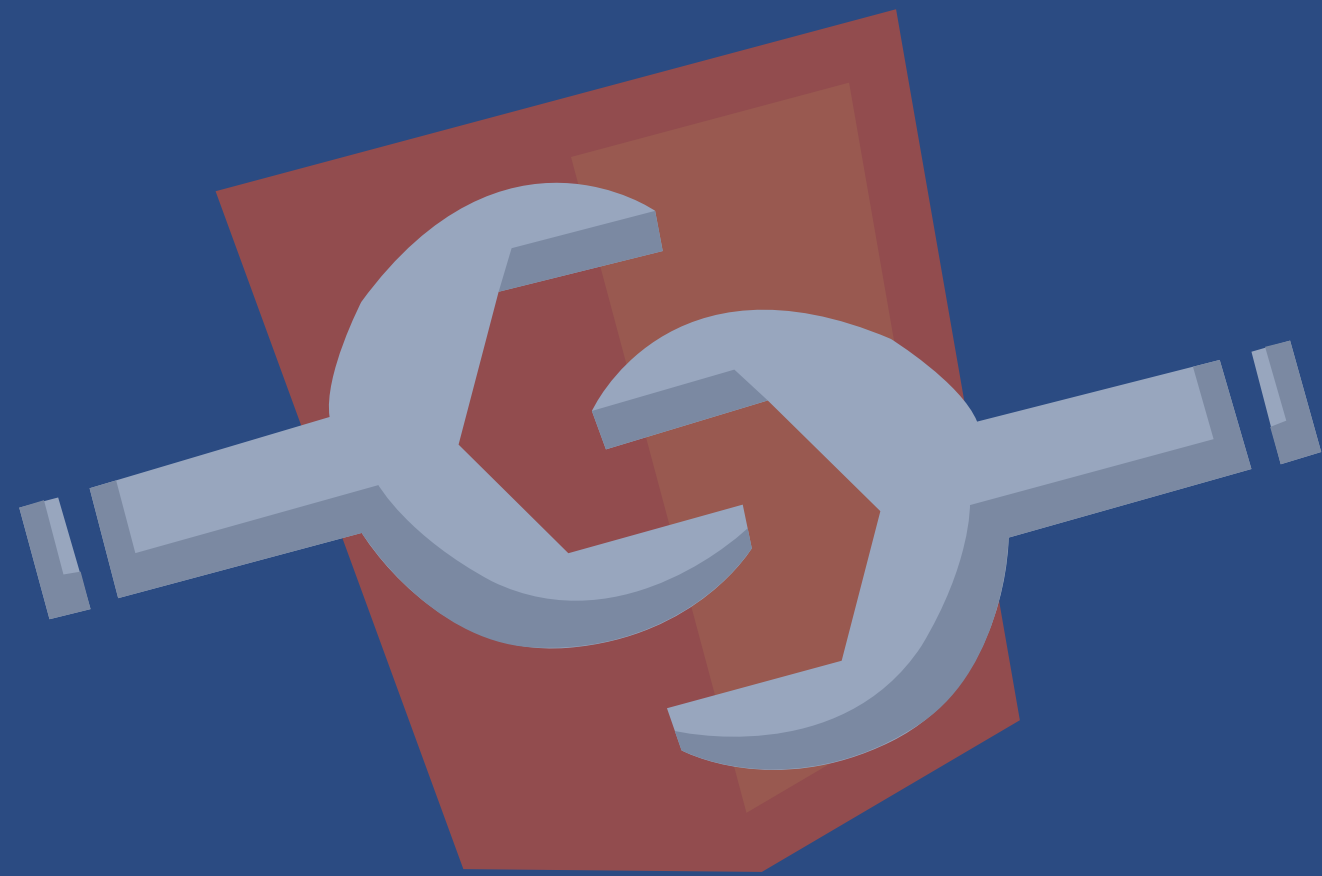


Our goal as a team was to design a CPU and implement the logic behind it using VHDL. Our CPU would include various components starting from Adders, ALU, MUX and all the way to Memory Instructions. After hours long research and discussions regarding the modern processor's design, we leaned towards implementing a 32-bit single cycle RISC-V core. And there we began our journey of designing our very own CPU!

Architecture

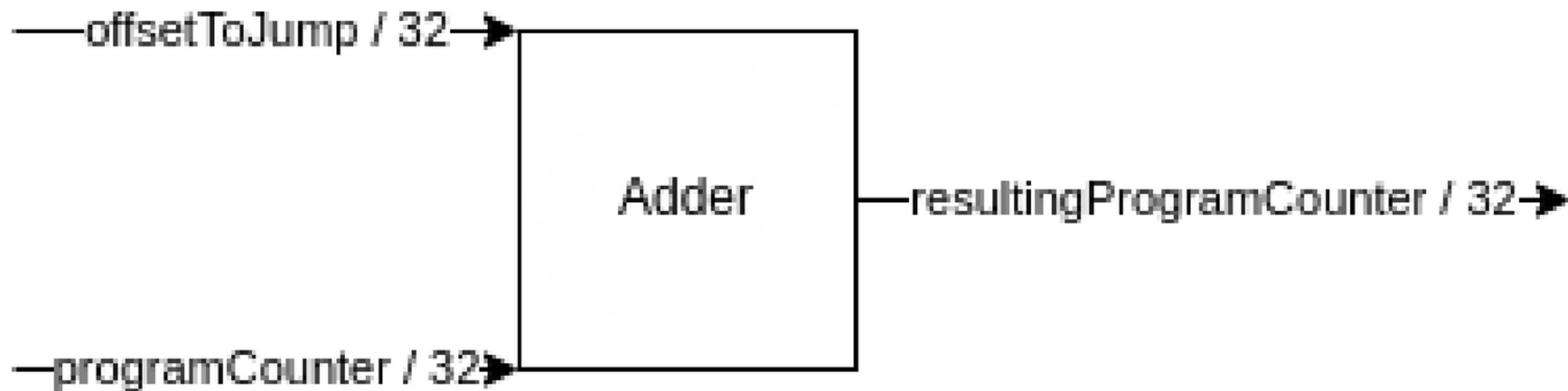
The architecture for the CPU apart from being RV32I (also known as Integer Architecture) was chosen to be Harvard Architecture. To put it simply, we had separate modules for data memory and instruction memory. Since we are using a single cycle CPU this architecture was chosen to be suitable.

Components



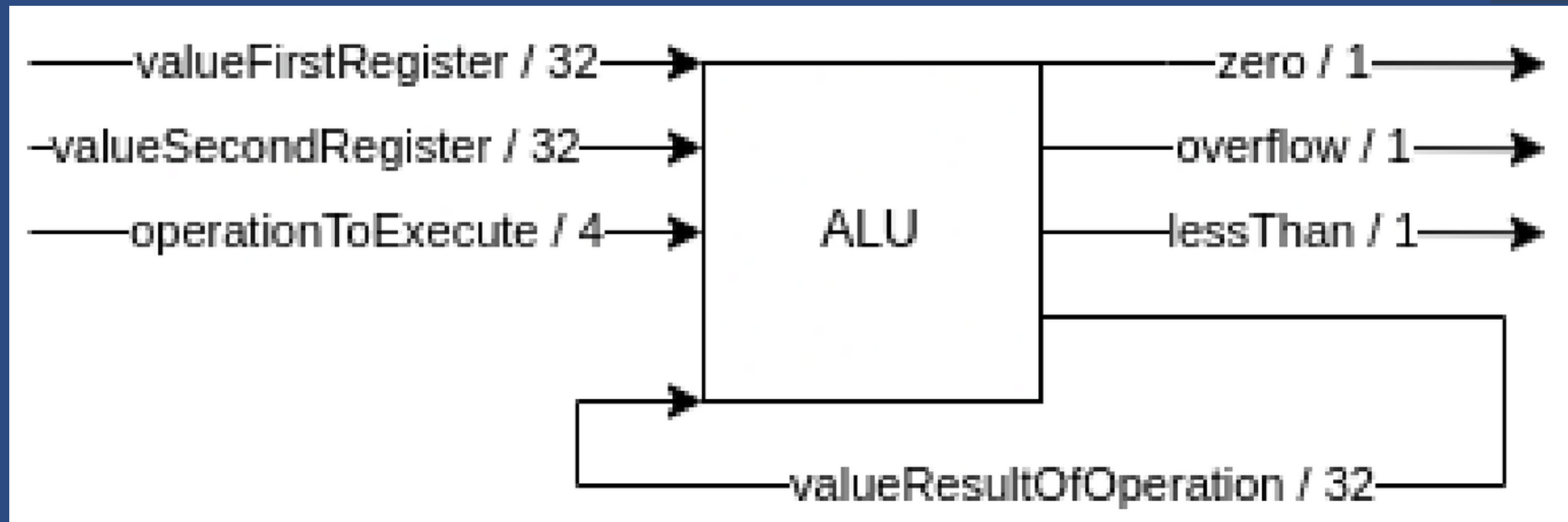
- Adder
- Arithmetic Logic Unit
- Data Memory
- Decoder
- Instruction Memory
- Multiplexer
- Register file
- Sign extender

Adder



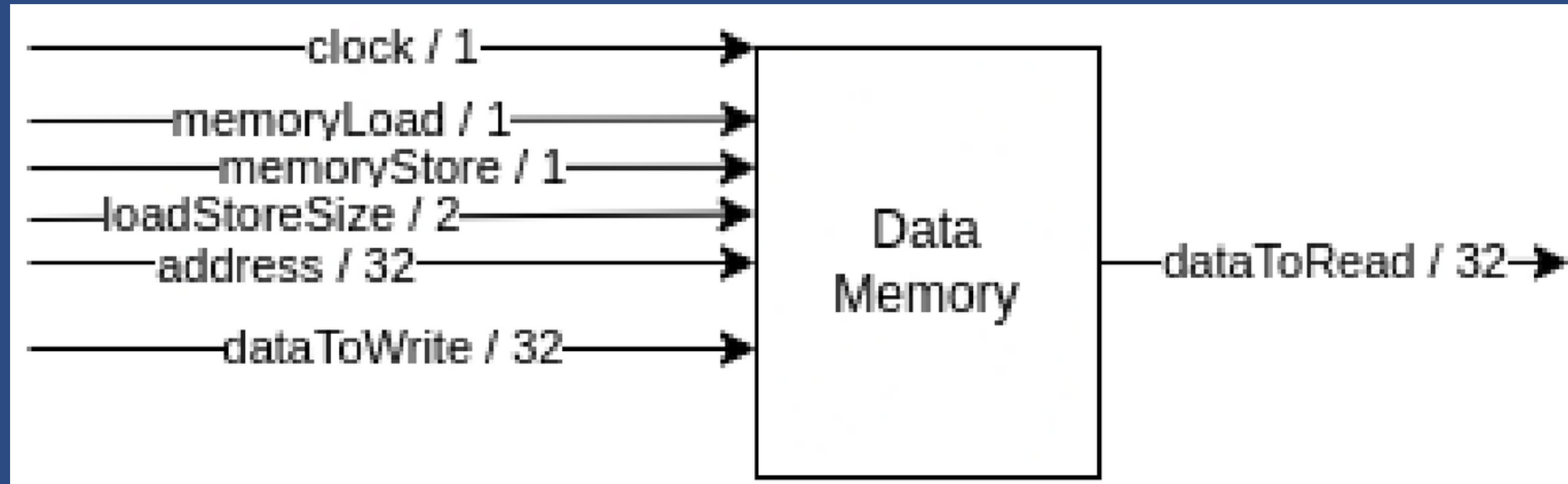
This module adds the program counter register to the offset to jump to. It allows us to use plus one as an increment which increments the actual counter by 32 bits. Since the addition is a rather lengthy operation, it's done using combinations logic, without the using clock.

Arithmetic Logic Unit

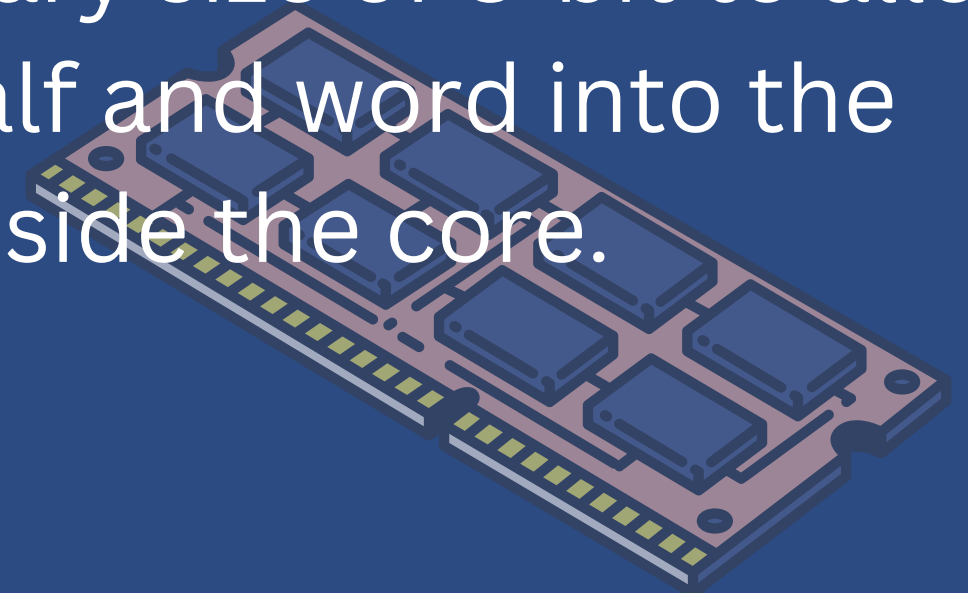


The ALU is used to carry the calculations on the registers. It takes 2 registers and a 4 bit instruction to carry. The output is a 32-bit result of operation and a flag for zero, overflow or less than. The circuit is implemented as sequential logic, since the rising clock edge can not be used the output updates on change of inputs rather than on each clock cycle.

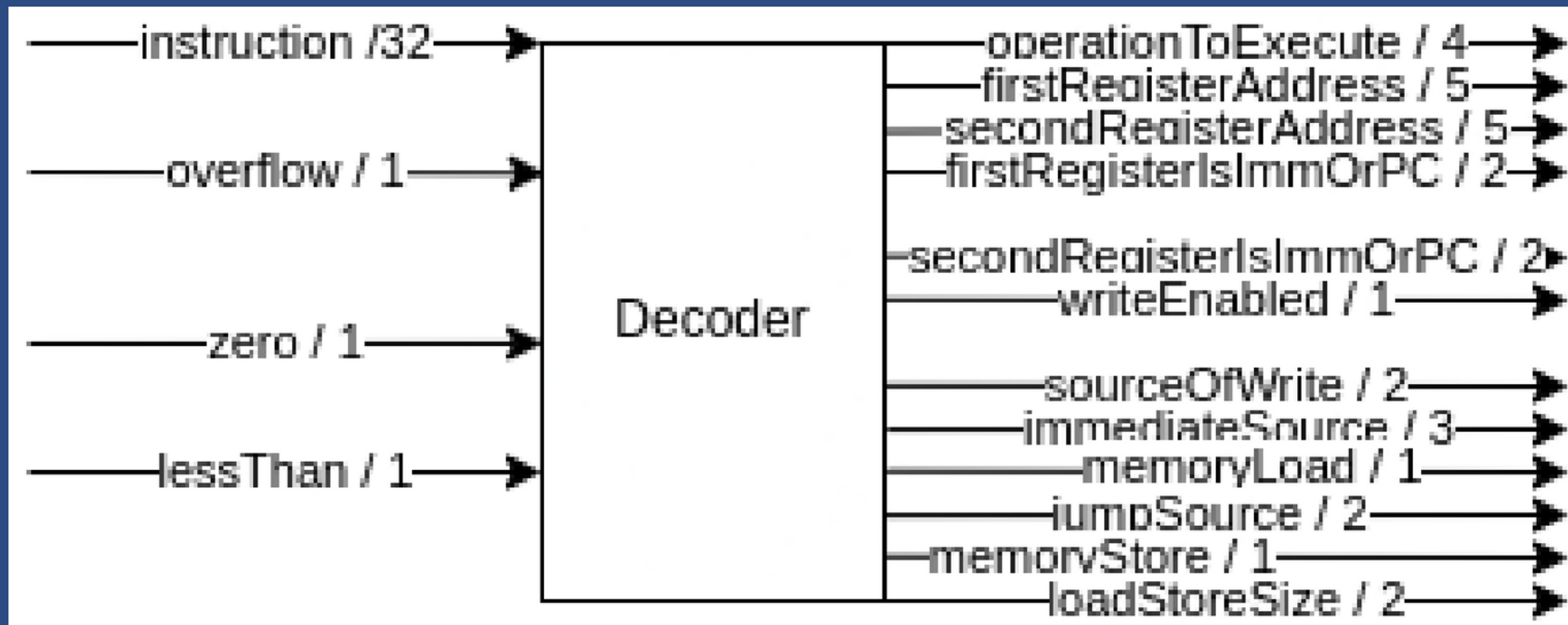
Data Memory



The data memory is implemented as an array of arbitrary size of 8-bit to allow the implementation of loading and storing byte, half and word into the registers. It can be used as cache memory inside the core.

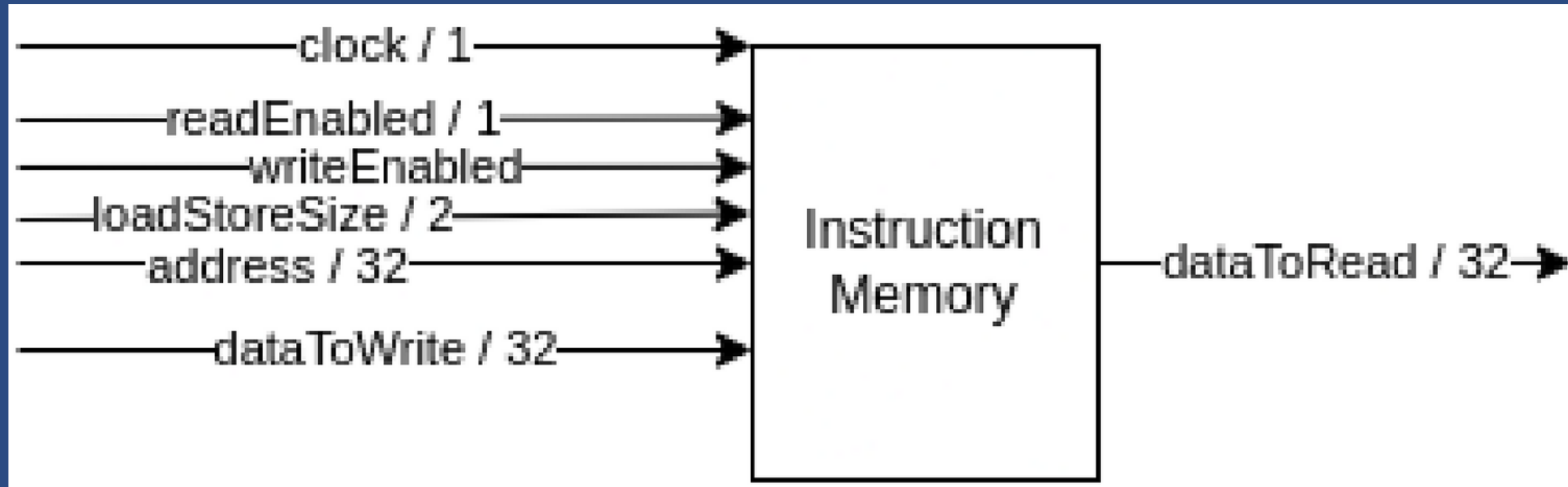


Decoder



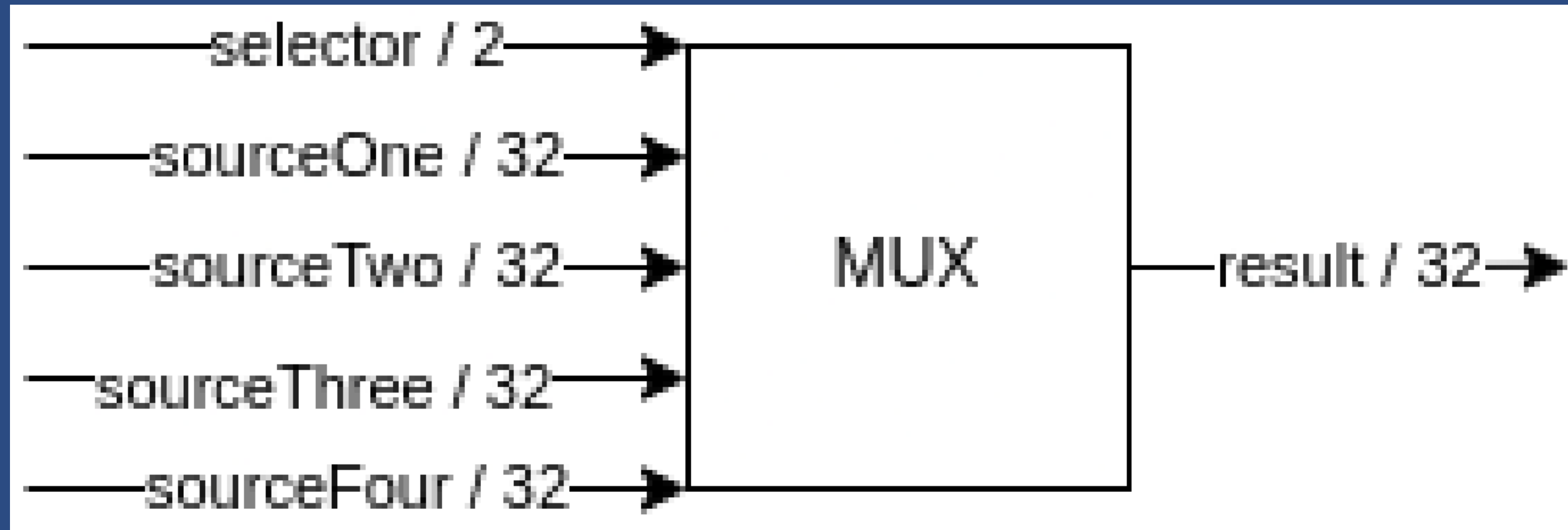
The decoder is the circuit which controls the data path. It takes the whole instruction decodes it to the flags needed to control the sources of the ALU and the operation to execute on them, the write ports of the memory and the register file as well as the decision to increment the program counter with a specified offset, i.e. jump.

Instruction Memory



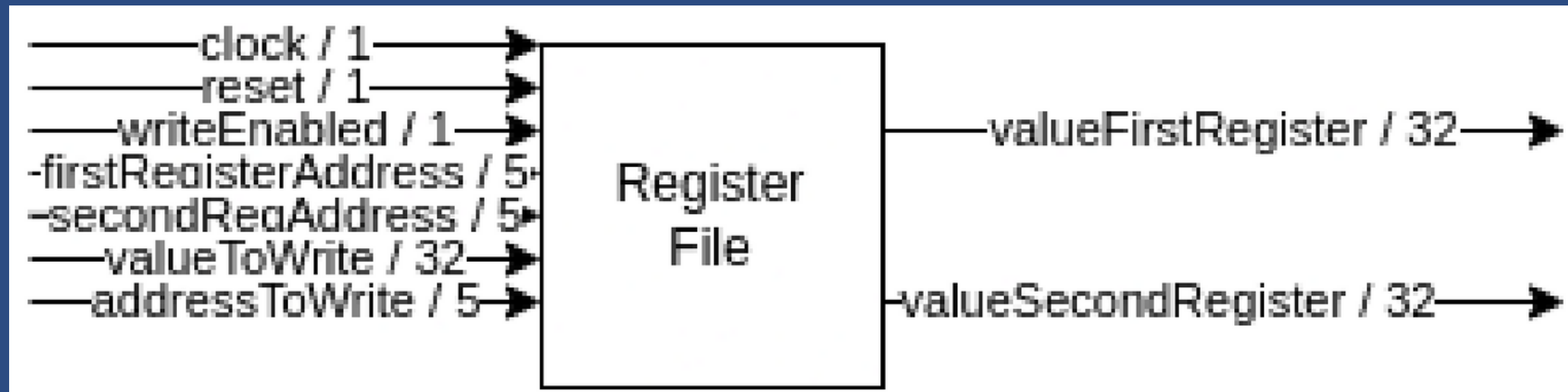
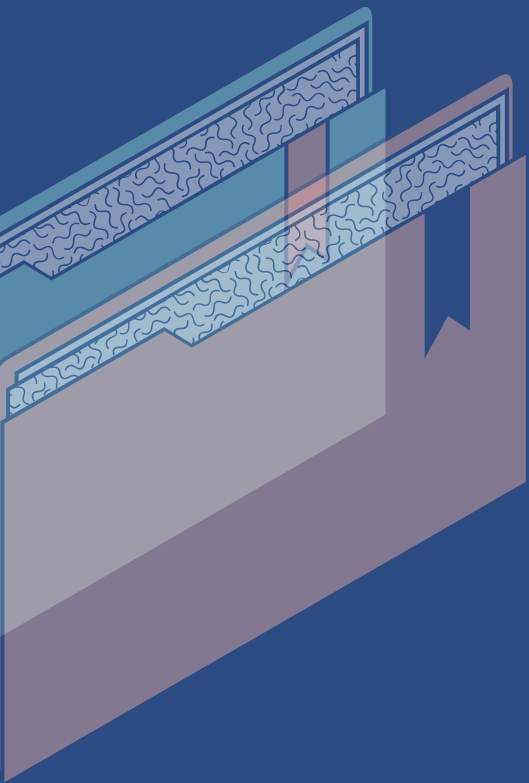
The instruction memory, much like the data memory, reads and writes on a rising clock edge. It is implemented as an array of 32 bit “std_logic_vector”. Instruction set and this memory is used for instructions only.

Multiplexer



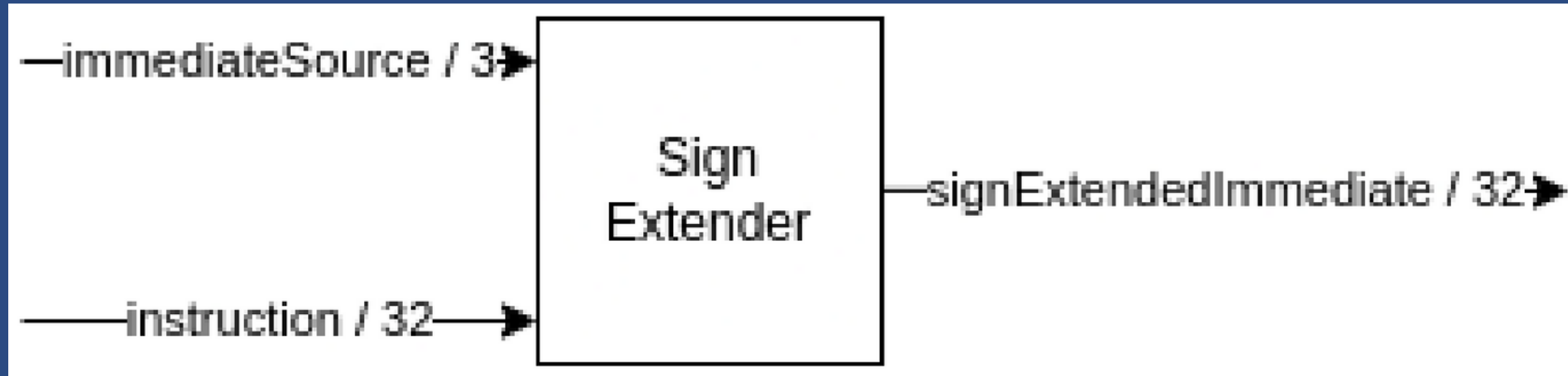
The multiplexer was used for selecting the source of jump for the program counter, the write port of the register file, the write port of the data memory. We used a 4-to-1 MUX and gave zero values to the unconnected ports.

Register File



The register file consists of 32 registers of XLEN size. The component is implemented as an array of "std_logic_vector" and is addressable by a 5-bit address. The register file consists of two read ports and one write port, as well as an address port for each of them. It consists of two processes one for writing to the register and one to read from the register file.

Sign Extender



The sign extender aims to build a 32-bit sign extended "immediate" from an 11-bit one. Depending on the type of instruction the position of the bits building the immediate varies. The instruction is used to add an arbitrary offset to the program counter unconditionally by providing a 20-bit immediate value which needs to be built to 32-bit before being added to the program counter.

CONCLUSION

To sum it up...

After hours of learning, researching, practicing, coding and testing we have fulfilled the goal of having a hands-on experience with VHDL design, simulations and programming using VHDL. Every member of our team have contributed his knowledge and expertise to accomplish the goals we set for the project.



Additional information, details and test cases can be found inside of the Project Report and Source files.

*Thank
you!*