

**Universidade Estadual do Oeste do Paraná (UNIOESTE)**  
**Centro de Engenharias e ciências exatas (CECE)**  
**Ciência da Computação - Compiladores**

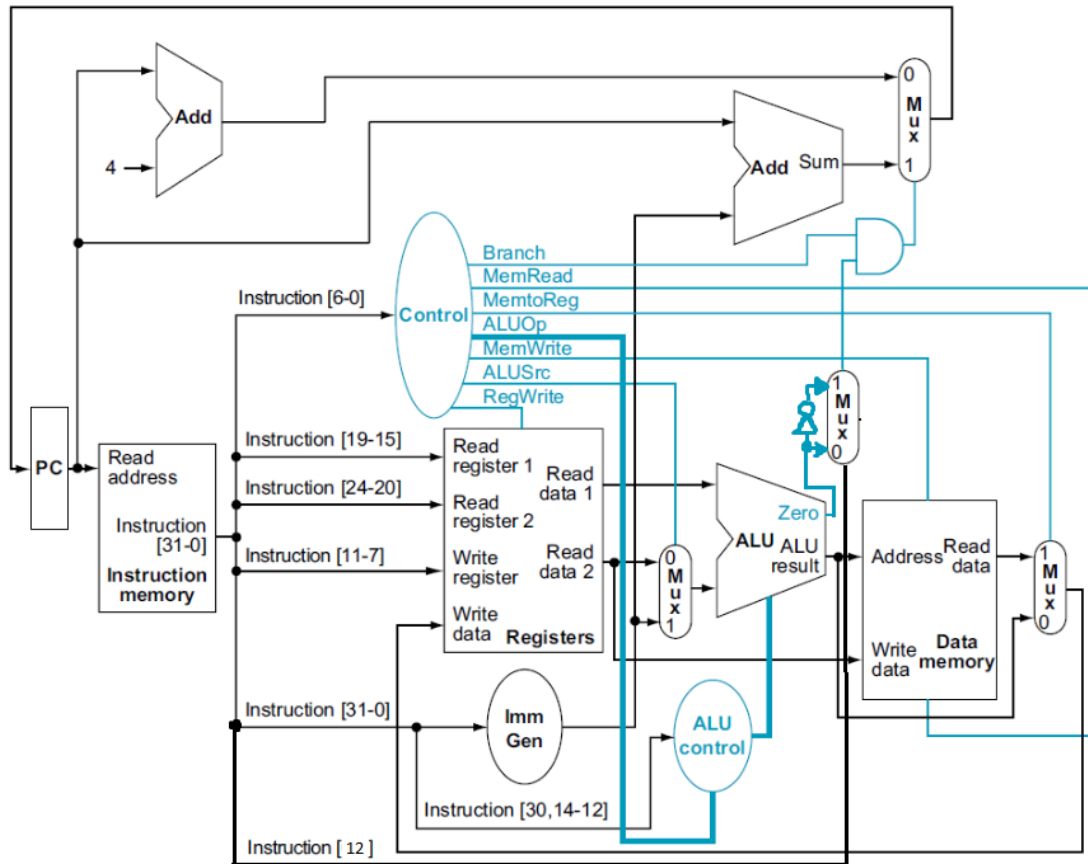
# **Máquina Virtual RISC-V Monociclo**

**Aluno:** Lucas Tomio Darim

**Professor:** Fabiana Frata Furlan Peres

## Implementação

Para a implementação do projeto, foi utilizada como base a via de dados da arquitetura RISC-V monociclo apresentada no livro de Patterson e Hennessy. A implementação foi feita em C++, onde foram criadas classes para cada unidade que compõe o processador, além de classes para representar o processador e a máquina. A figura 01 mostra a via de dados utilizada, que é capaz de realizar as instruções add, sub, and, or, lw, sw, beq e bne.



**Figura 01: Via de Dados**

Para melhor organização e modularização do código, cada unidade que compõe o processador foi separada em classes diferentes na implementação. Além disso, foram criadas classes para representar o processador e a máquina. A classe do processador contém os objetos que formam o processador e é responsável por fazer a integração entre eles. Já a classe da máquina apenas carrega e decodifica as instruções do arquivo de texto e envia o sinal para o processador fazer um clock.

Ao serem lidas do arquivo de texto as instruções são transformadas para valores inteiros, assim facilitando a passagem como parâmetro para cada uma das classes. No caso de precisar de apenas alguns dos bits é utilizado shift para o deslocamento e and com uma máscara para obter os bits necessários.

## Instruções para o uso

Dentre os arquivos do projeto, há um arquivo "Makefile" que contém todas as especificações para a compilação do código-fonte. Para compilar o código-fonte, basta utilizar o comando "make" estando no diretório do projeto. O compilador utilizado é o clang++ e o resultado é um executável chamado "riscv".

```
(lucas@caladan)~[~/Documents/RISC-V-Monociclo]
$ ls
adder.cpp      control.cpp    instructionMemory.cpp  multiplexer.cpp  registers.cpp
adder.hpp      control.hpp    instructionMemory.hpp  multiplexer.hpp  registers.hpp
aluControl.cpp dataMemory.cpp machine.cpp          processor.cpp     testes
aluControl.hpp dataMemory.hpp machine.hpp          processor.hpp
alu.cpp        immGen.cpp     main.cpp            programCounter.cpp
alu.hpp        immGen.hpp     Makefile            programCounter.hpp

(lucas@caladan)~[~/Documents/RISC-V-Monociclo]
$ make
clang++ -g -Wno-everything ./aluControl.cpp ./instructionMemory.cpp ./programCounter.cpp ./r
egisters.cpp ./control.cpp ./machine.cpp ./immGen.cpp ./processor.cpp ./dataMemory.cpp ./mul
tiplexer.cpp ./adder.cpp ./main.cpp ./alu.cpp -o "riscv"

(lucas@caladan)~[~/Documents/RISC-V-Monociclo]
$ ls
adder.cpp      control.cpp    instructionMemory.cpp  multiplexer.cpp  registers.cpp
adder.hpp      control.hpp    instructionMemory.hpp  multiplexer.hpp  registers.hpp
aluControl.cpp dataMemory.cpp machine.cpp          processor.cpp     riscv
aluControl.hpp dataMemory.hpp machine.hpp          processor.hpp     testes
alu.cpp        immGen.cpp     main.cpp            programCounter.cpp
alu.hpp        immGen.hpp     Makefile            programCounter.hpp
```

Figura 02: Compilação do código

Com o executável, podemos executar o programa com o seguinte comando: `./riscv <arquivo/de/instruções>`. O parâmetro "`<arquivo/de/instruções>`" deve ser substituído pelo caminho até o arquivo que contém as instruções.

```
(lucas@caladan)~[~/Documents/RISC-V-Monociclo]
$ ./riscv testes/teste1.txt

Trabalho de Organização e Arquitetura de Computadores
Máquina Virtual RISC-V Monociclo
Lucas Tomio Darim

Digite n para próximo ciclo ou q para sair.n
=====
```

Figura 03: Execução do programa

Ao executar o programa, o usuário pode digitar "n" para que a máquina execute o próximo ciclo de clock ou "q" para sair do programa. A cada ciclo de clock, é impresso na tela o output de cada uma das unidades do processador na sequência em que estão sendo executadas para que seja possível entender o

funcionamento da máquina virtual. Além disso, são impressos todos os registradores e posições da memória que contém um valor armazenado diferente de zero.

```
=====
ProgramCounter: 44
InstructionMemory: 12970291
Adder1: 48
Control:
    Instrução do Tipo R
    Branch : 0
    MemRead : 0
    MemToReg: 0
    ALUOP : 2
    MemWrite: 0
    ALUSrc : 0
    RegWrite: 1
Registers Read:
    ReadData1: 3
    ReadData2: 10
ImmGen:
    Output: 4
Mux1: 10
ALUControl: 1
ALU:
    Output: 11
    Zero : 0
Adder2: 48
Mux2: 0
Mux3: 48
DataMemory:
Mux4: 11
Registers Write:
    Register: 0
    Data : 11
-----Registradores-----
reg(5) = 3
reg(6) = 10
reg(10) = 20
reg(11) = 3
reg(12) = 10
reg(14) = 13
reg(15) = 7
reg(16) = -7
reg(17) = 2
reg(18) = 11
-----Memoria-----
Mem[20] = 3
Mem[24] = 10
=====
```

Figura 04: Exemplo de saída do programa