# MICROARQUITETURA SINGLE CYCLE EM FPGA

**Projeto final Laboratório de Sistemas Digitais**

Víctor César Teixeira Santos

# SUMÁRIO

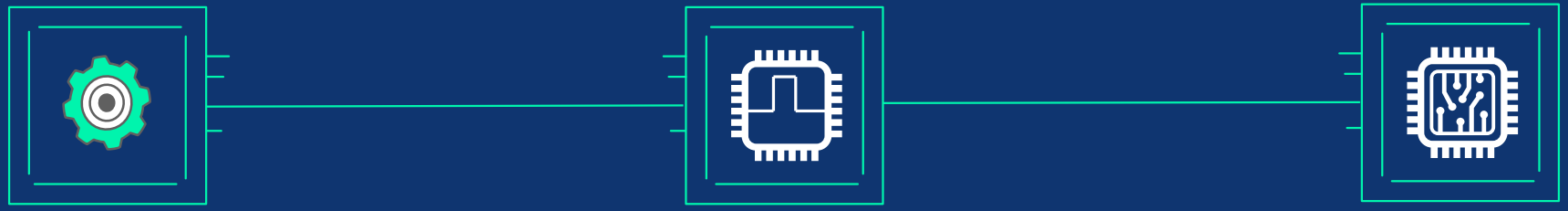
0x003


# OBJETIVOS

Planejamento e resultados esperados

0x004

# OBJETIVOS



Projeto de uma Microarquitetura Single Cycle de um processador MIPS em VHDL. Simulação funcional

Obtenção e análise do circuito prático. Análise de clock.

Implementação em uma FPGA

0x005

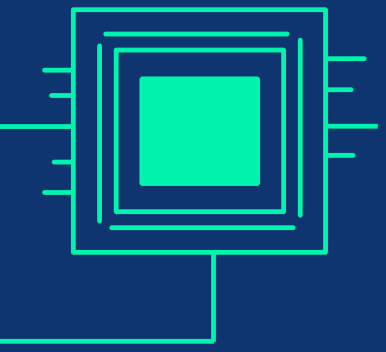

# INTRODUÇÃO

Conceitos necessários ao projeto.

# MIPS

- MIPS (Microprocessor without Interlocked Pipeline Stages) é um processador criado em 1984 que utiliza a arquitetura de conjunto de instruções MIPS;
- Processador do tipo RISC (Reduced Instruction Set Computer);
- Há 111 instruções com 32 bits cada.
- A partir da instrução em assembly há a conversão em código de máquina, entendível ao computador;

Exemplo de instrução:

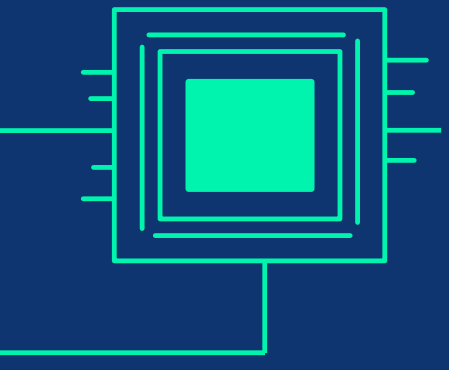


```
lw $s0,40($t2)
```

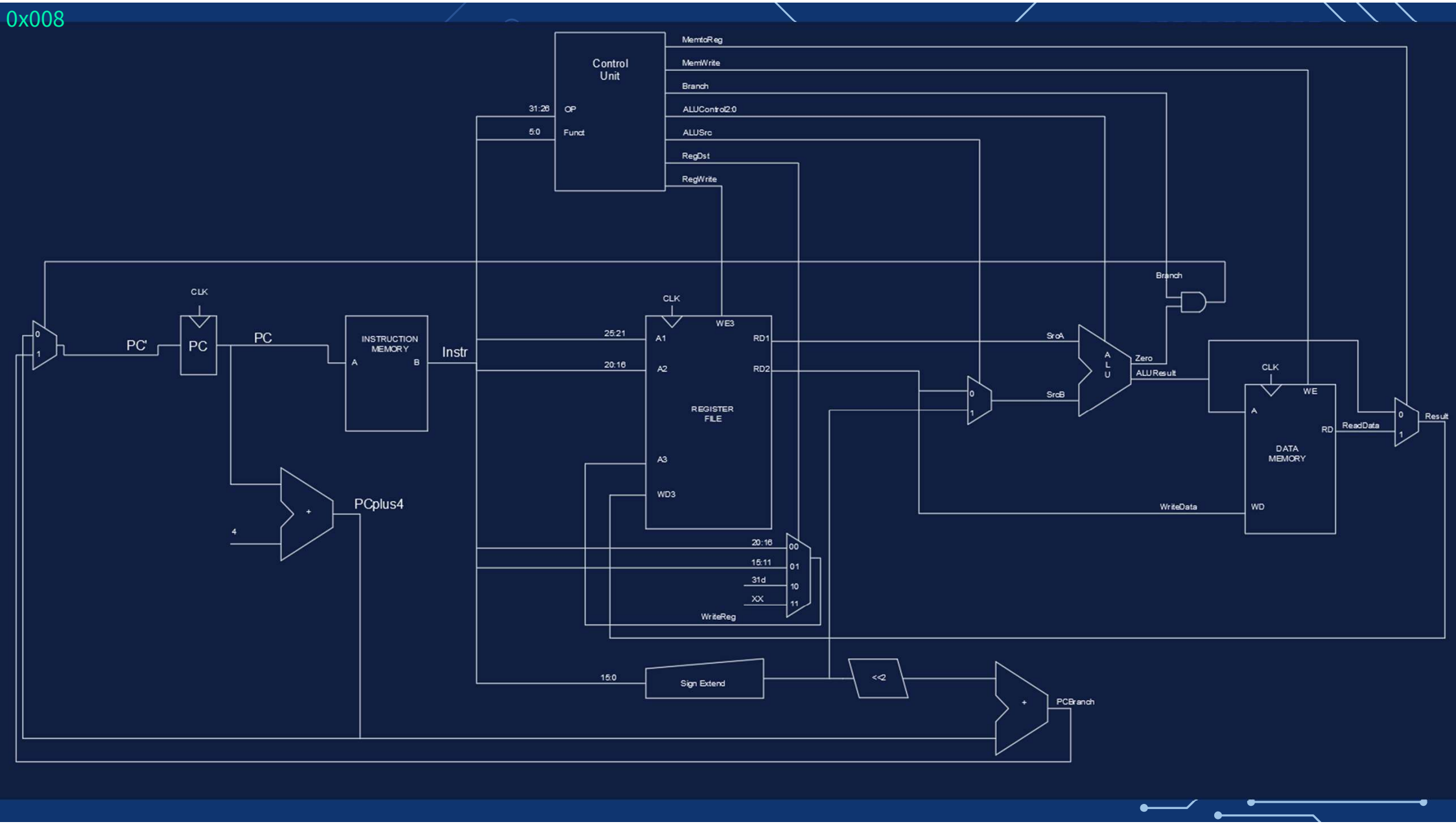
Mnemônico
Registrador de destino
Offset
Endereço base

# MICROARQUITETURA

- Ligação entre a lógica e a arquitetura através do arranjo de blocos lógicos como ALUs, Registradores, Memórias, etc.
- Também denominada como Organização de Computadores;
- Arquitetura MIPS pode ter diferentes microarquiteturas: *Single Cycle, Multicycle, Single Cycle* com *Pipeline;*
- *Single Cycle* define que todas as instruções são executadas por completo em um único ciclo de clock

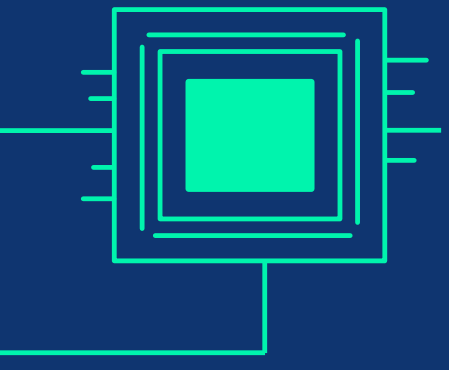


0x008


Control Unit
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
31:28
OP
5:0
Funct
CLK
PC'
PC
PC
INSTRUCTION MEMORY
A
B
Instr
CLK
WE3
25:21
A1
RD1
20:16
A2
RD2
REGISTER FILE
A3
WD3
SrcA
SrcB
A L U
Zero
ALUResult
Branch
CLK
WE
A
RD
DATA MEMORY
WD
ReadData
WriteData
Result
PCplus4
4
20:16
15:11
31d
XX
00
01
10
11
WriteReg
15:0
Sign Extend
<<2
PCBranch

# VHDL E FPGA

- VHDL: **V**HSIC (Very High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage: Linguagem para descrever sistemas digitais e para realizar testes funcionais;
- FPGA: Circuito integrado cuja principal característica é poder ser (re)programado para uso específico após a sua manufatura.
- Descrição em VHDL deve ser sintetizada por um software e posteriormente carregada em uma placa FPGA;



0x00A


# MÉTODOS

Realização do projeto

0x00B

# ABSTRAÇÃO E DESCRIÇÃO

01

## ESQUEMÁTICO

Captura do esquemático do nível de abstração mais baixo ao mais alto

02

## DESCRIÇÃO DOS BLOCOS BÁSICOS

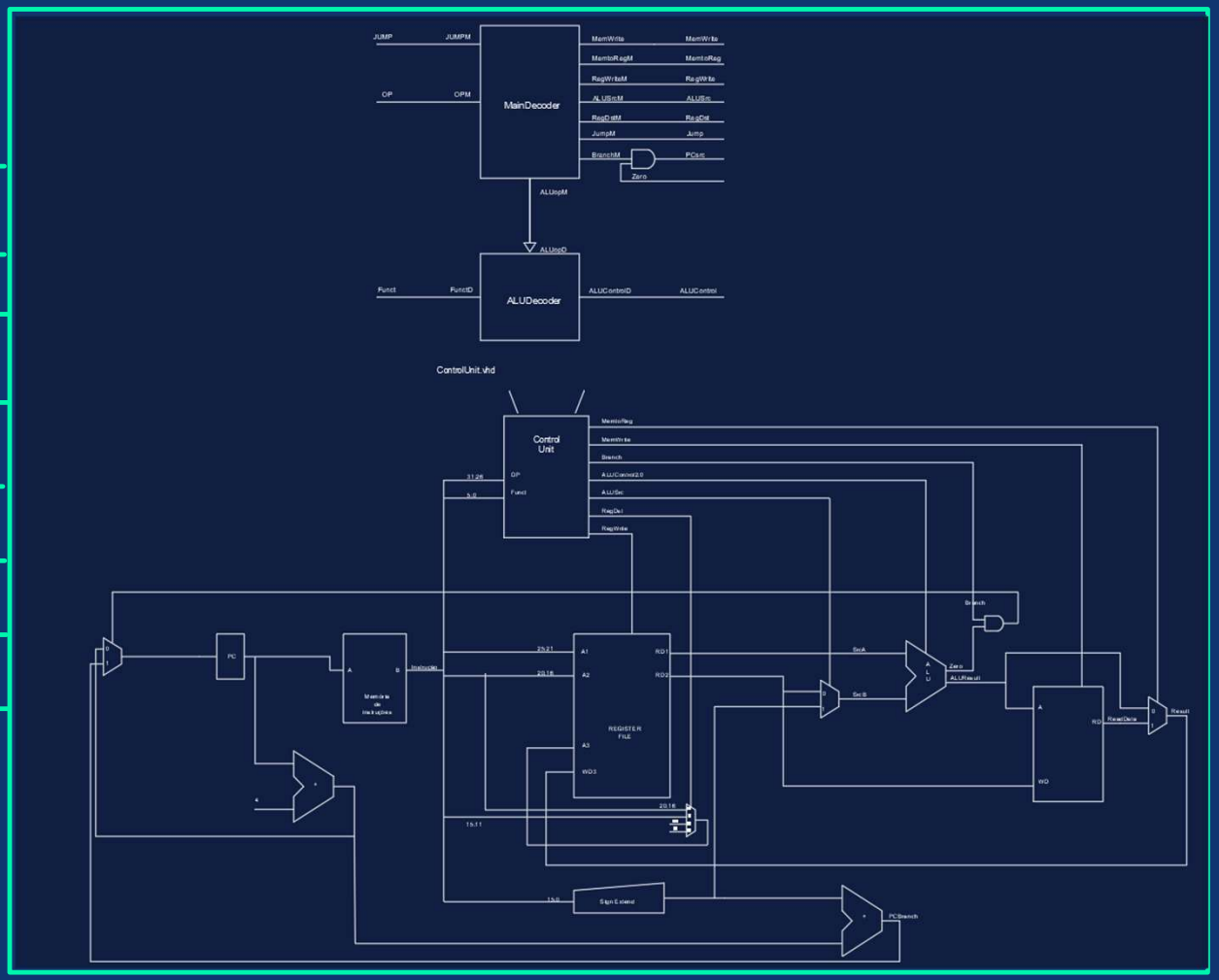
Descrição feita em VHDL e Testbench com um número grande de testes

03

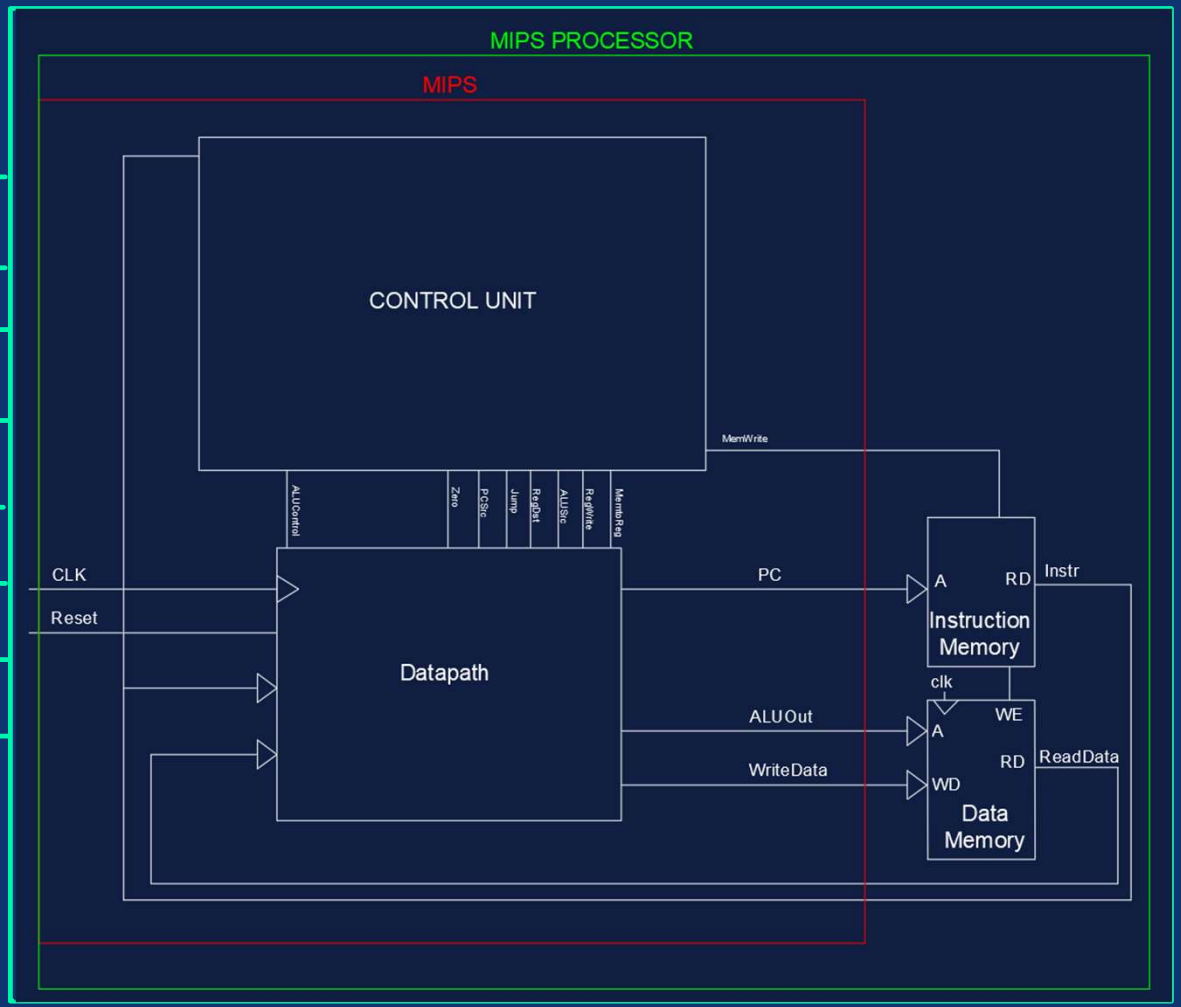
## TESTE DESCRIÇÃO

A cada descrição de bloco um teste realizado

# PRIMEIRO ESQUEMÁTICO



# SEGUNDO ESQUEMÁTICO



0x00E

# TESTES E IMPLEMENTAÇÃO

04

## UNIÃO DOS BLOCOS

A união dos blocos básicos aumenta o nível de abstração até formar toda a microarquitetura

05

## TESTE UNIDADE

A cada aumento do nível de abstração, mais um teste realizado

06

## IMPLEMENTAÇÃO FPGA

Com o código correto, simulado e sintetizado, o circuito era carregado na FPGA

# TESTBENCH

Para os blocos mais simples, como MUX, Program Counter, Register File, etc. Utilizou-se valores “aleatórios” em um arquivo “.txt” de leitura e comparação;

ALU > tb_ALU.vhd

```vhdl
use ieee.std_logic_textio.all;
use std.textio.all;

entity tb_ALU is
end tb_ALU;

architecture test of tb_ALU is
component ALU is
port (
    ALUControl  : in    std_logic_vector(2 downto 0);
    SrcA        : in    std_logic_vector(31 downto 0);
    SrcB        : in    std_logic_vector(31 downto 0);
    ZEROFlag    : out   std_logic;
    ALUResult   : out   std_logic_vector(31 downto 0)
);
end component;
    -- Clock period definitions
    constant PERIOD     : time := 20 ns;
    constant DUTY_CYCLE : real := 0.5;
    constant OFFSET     : time := 5 ns;

    file    inputs_data_ALU   : text open read_mode  is "inputALU.txt";
    file    data_compare      : text open read_mode  is "inputcompare.txt";
    file    outputs_data      : text open write_mode is "output.txt";
    file    outputs_data_comp : text open write_mode is "outputdata_comp.txt";
```

0x010

# ARQUIVOS

## Entrada entidade

ALU > inputALU.txt

```
000
00000000
00000010
001
00000000
00000010
010
00000000
00000010
110
00000000
00000010
111
00000000
00000010
000
00000001
00000010
001
00000001
00000010
010
00000001
00000010
110
00000001
00000010
111
00000001
00000010
```

## Saída entidade

ALU > output.txt

```
ALUresult
00000000000000000000000000000000
ZEROFlag
1
------------------------
ALUresult
00000000000000000000000000010000
ZEROFlag
0
------------------------
ALUresult
00000000000000000000000000010000
ZEROFlag
0
------------------------
ALUresult
11111111111111111111111111110000
ZEROFlag
0
------------------------
ALUresult
00000000000000000000000000000001
ZEROFlag
0
------------------------
ALUresult
00000000000000000000000000000000
ZEROFlag
1
```

## Entrada comparação

ALU > inputcompare.txt

```
00000000000000000000000000000000
1
00000000000000000000000000010000
0
00000000000000000000000000010000
0
11111111111111111111111111110000
0
00000000000000000000000000000001
0
00000000000000000000000000000000
1
00000000000000000000000000010001
0
00000000000000000000000000010001
0
11111111111111111111111111110001
0
00000000000000000000000000000001
0
00000000000000000000000000000000
1
00000000000000000000000000010010
0
00000000000000000000000000010010
0
11111111111111111111111111110010
0
00000000000000000000000000000001
```

## Saída comparação

ALU > outputdata_comp.txt

```
ALU Result
GOOD
Zero Flag
Good
------------------------
ALU Result
GOOD
Zero Flag
Good
------------------------
ALU Result
GOOD
Zero Flag
Good
------------------------
ALU Result
GOOD
Zero Flag
Good
------------------------
ALU Result
GOOD
Zero Flag
Good
------------------------
ALU Result
GOOD
Zero Flag
Good
------------------------
```

# TESTE UNIDADE

Em unidades em que a abstração é maior (DataPath, MIPS, MIPS Processor), foi utilizado um código teste simples de um conjunto de instruções.
Os valores em binário foram retirados do programa MARS;

| Address | Code | Basic | Source |
|---|---|---|---|
| 0x00400000 | 0x20160064 | addi $22,$0,100 | 1 addi $s6, $0, 100 |
| 0x00400004 | 0xac160028 | sw $22,40($0) | 2 sw $s6, 40($0) |
| 0x00400008 | 0x8c0a0028 | lw $10,40($0) | 3 lw $t2, 40($0) |
| 0x0040000c | 0x02ca9820 | add $19,$22,$10 | 4 add $s3, $s6, $t2 |
| 0x00400010 | 0x026aa022 | sub $20,$19,$10 | 5 sub $s4, $s3, $t2 |
| 0x00400014 | 0x0276a824 | and $21,$19,$22 | 6 and $s5, $s3, $s6 |
| 0x00400018 | 0x02b4b825 | or $23,$21,$20 | 7 or $s7, $s5, $s4 |
| 0x0040001c | 0x02758820 | add $17,$19,$21 | 8 add $s1, $s3, $s5 |
| 0x00400020 | 0xac110028 | sw $17,40($0) | 9 sw $s1, 40($0) |
| 0x00400024 | 0x8c080028 | lw $8,40($0) | 10 lw $t0, 40($0) |
| 0x00400028 | 0x0000000c | syscall | 11 syscall |

0x012


# RESULTADOS

Arquivos gerados, simulação funcional e simulação na placa

# DATAPATH

DataPath > DataPath.vhd

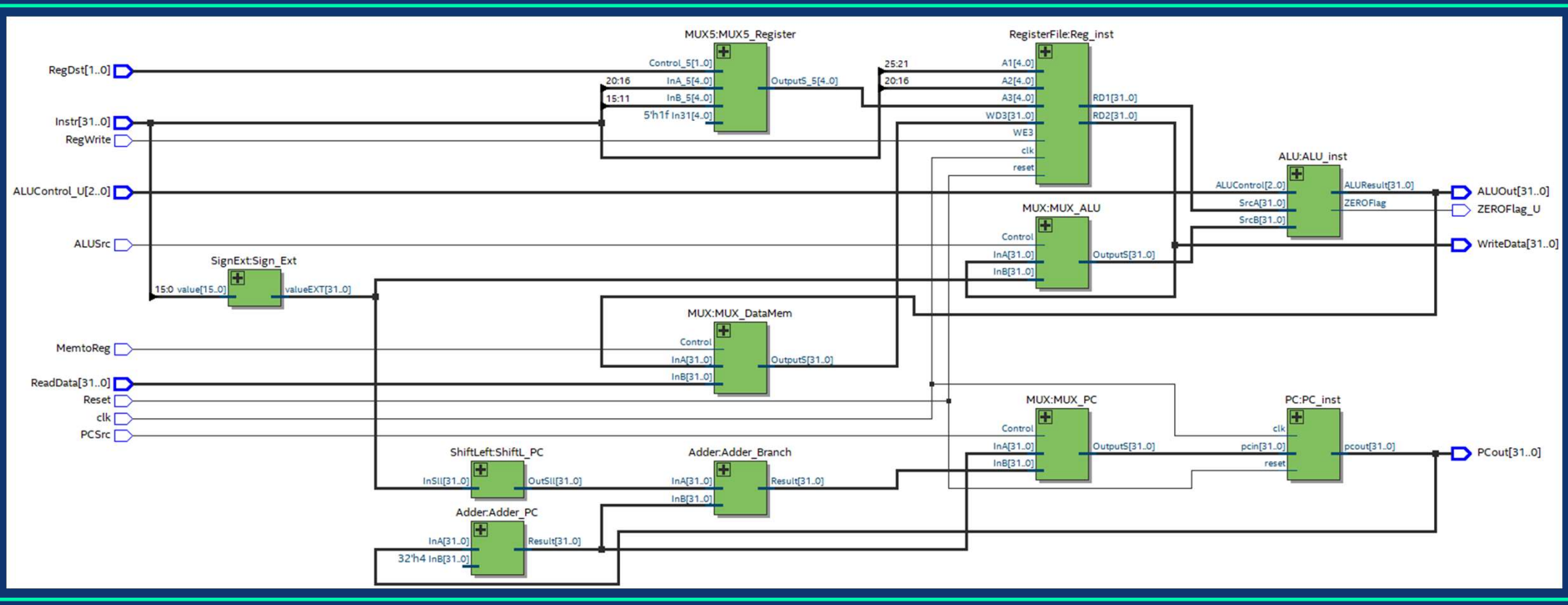
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5    entity DataPath is port(
6           Instr        : in std_logic_vector(31 downto 0);
7           Reset        : in std_logic;
8           clk          : in std_logic;
9           ReadData     : in std_logic_vector(31 downto 0);
10          RegWrite     : in std_logic;
11          RegDst       : in std_logic_vector(1 downto 0);
12          ALUSrc       : in std_logic;
13          MemtoReg     : in std_logic;
14          PCSrc        : in std_logic;
15          ALUControl_U : in std_logic_vector(2 downto 0);
16          ZEROFlag_U   : out std_logic;
17          PCout        : out std_logic_vector(31 downto 0);
18          ALUOut       : out std_logic_vector(31 downto 0);
19          WriteData    : out std_logic_vector(31 downto 0)
20  );
21  end DataPath;
```

O Caminho de Dados é onde será instanciado os componentes como *Register File, Program Counter, ALU,* etc.

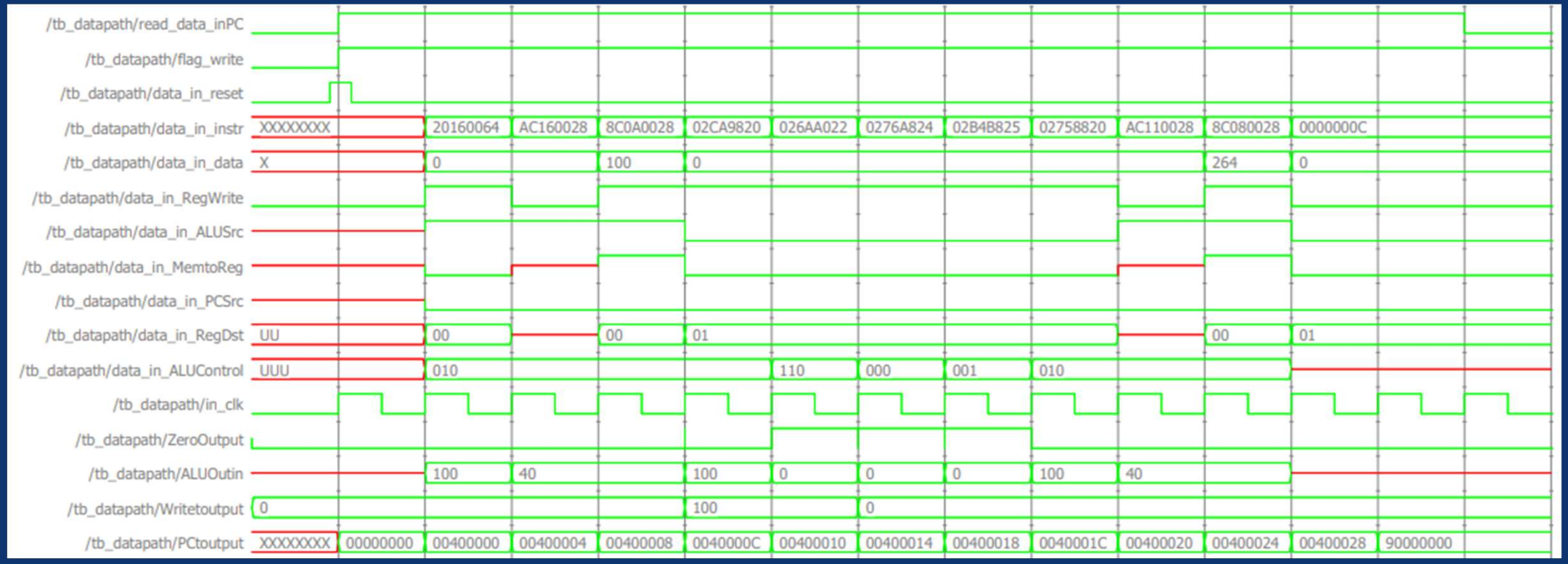
É a parte responsável por executar as operações sobre os dados recebidos da Memória de Dados.

As operações são definidas a partir da instrução sendo executada, e o controle dos components é feito a partir dos sinais recebidos da Unidade de Controle.

# DATAPATH


MUX5:MUX5_Register
RegisterFile:Reg_inst
RegDst[1..0]
Instr[31..0]
RegWrite
ALUControl_U[2..0]
ALUSrc
MemtoReg
ReadData[31..0]
Reset
clk
PCSrc
Control_5[1..0]
InA_5[4..0]
InB_5[4..0]
5'h1f In31[4..0]
OutputS_5[4..0]
20:16
15:11
25:21
20:16
A1[4..0]
A2[4..0]
A3[4..0]
WD3[31..0]
WE3
clk
reset
RD1[31..0]
RD2[31..0]
ALU:ALU_inst
ALUControl[2..0]
SrcA[31..0]
SrcB[31..0]
ALUResult[31..0]
ZEROFlag
ALUOut[31..0]
ZEROFlag_U
WriteData[31..0]
MUX:MUX_ALU
Control
InA[31..0]
InB[31..0]
OutputS[31..0]
SignExt:Sign_Ext
15:0 value[15..0]
valueEXT[31..0]
MUX:MUX_DataMem
Control
InA[31..0]
InB[31..0]
OutputS[31..0]
MUX:MUX_PC
Control
InA[31..0]
InB[31..0]
OutputS[31..0]
PC:PC_inst
clk
pcin[31..0]
reset
pcout[31..0]
PCout[31..0]
ShiftLeft:ShiftL_PC
InSll[31..0]
OutSll[31..0]
Adder:Adder_Branch
InA[31..0]
InB[31..0]
Result[31..0]
Adder:Adder_PC
InA[31..0]
32'h4 InB[31..0]
Result[31..0]

# DATAPATH


/tb_datapath/read_data_inPC
/tb_datapath/flag_write
/tb_datapath/data_in_reset
/tb_datapath/data_in_instr
XXXXXXXX
20160064
AC160028
8C0A0028
02CA9820
026AA022
0276A824
02B4B825
02758820
AC110028
8C080028
0000000C
/tb_datapath/data_in_data
X
0
100
0
264
0
/tb_datapath/data_in_RegWrite
/tb_datapath/data_in_ALUSrc
/tb_datapath/data_in_MemtoReg
/tb_datapath/data_in_PCSrc
/tb_datapath/data_in_RegDst
UU
00
00
01
00
01
/tb_datapath/data_in_ALUControl
UUU
010
110
000
001
010
/tb_datapath/in_clk
/tb_datapath/ZeroOutput
/tb_datapath/ALUOutin
100
40
100
0
0
0
100
40
/tb_datapath/Writetoutput
0
100
0
/tb_datapath/PCtoutput
XXXXXXXX
00000000
00400000
00400004
00400008
0040000C
00400010
00400014
00400018
0040001C
00400020
00400024
00400028
90000000

# MAIN DECODER

Control Unit > MainDecoder.vhd

```
6   library IEEE;
7   use IEEE.std_logic_1164.all;
8   use IEEE.numeric_std.all;
9
10  entity maindecoder is
11      port(
12              OPM         : in std_logic_vector(5 downto 0);
13              MemToRegM   : out std_logic;
14              MemWriteM   : out std_logic;
15              BranchM     : out std_logic;
16              AluSrcM     : out std_logic;
17              RegDstM     : out std_logic_vector(1 downto 0);
18              RegWriteM   : out std_logic;
19              jumpM       : out std_logic;
20              ALUopM      : out std_logic_vector(1 downto 0)
21      );
22  end maindecoder;
```

O *Main Decoder* faz parte da *Control Unit* e é responsável por gerar os sinais de controle para os componentes do *Data Path* MENOS DA ALU a partir do recebimento do *Opcode* da Instrução.

A partir dele pode-se estabelecer as instruções que o MIPS poderá suportar.

# ALU DECODER

Control Unit > ALUDecoder.vhd

```
8   library IEEE;
9   use IEEE.std_logic_1164.all;
10  use IEEE.numeric_std.all;
11
12  entity ALUDecoder is
13      port(   FunctD      : in std_logic_vector(5 downto 0);
14              ALUopD      : in std_logic_vector(1 downto 0);
15              ALUControlD : out std_logic_vector(2 downto 0)
16      );
17  end ALUDecoder;
18
19  architecture ALUDecARCH of ALUDecoder is
20  begin
21      process (all)
22      begin
23          case ALUopD is
24              when "00" =>
25              ALUControlD <= "010"; --add
26              when "01" =>
27              ALUControlD <= "110"; --subtract
```

O *ALU Decoder* também faz parte da *Control Unit* e é responsável por gerar os sinais de controle exclusivamente para a ALU.

O valor da saída pode ser estabelecido apenas pela leitura do ALUOP vindo diretamente do *Main Decoder* ou da leitura do campo *Funct* da instrução.

# CONTROL UNIT

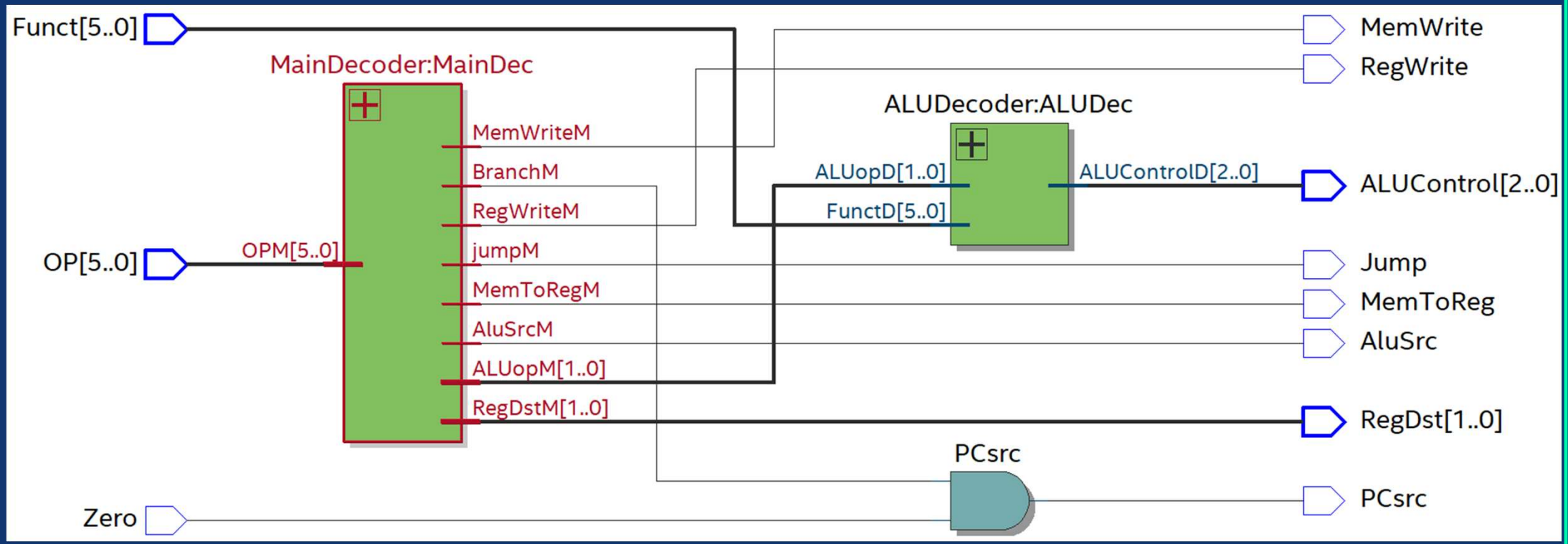
Control Unit > ControlUnit.vhd

```
1   library IEEE;
2   use ieee.std_logic_1164.all;
3
4   entity ControlUnit is
5       port(
6           OP          : in std_logic_vector (5 downto 0);
7           Funct       : in std_logic_vector (5 downto 0);
8           MemToReg    : out std_logic;
9           MemWrite    : out std_logic;
10          Zero        : in std_logic;
11          PCsrc       : out std_logic;
12          ALUControl  : out std_logic_vector(2 downto 0);
13          AluSrc      : out std_logic;
14          RegDst      : out std_logic_vector(1 downto 0);
15          RegWrite    : out std_logic;
16          Jump        : out std_logic
17      );
18  end ControlUnit;
```

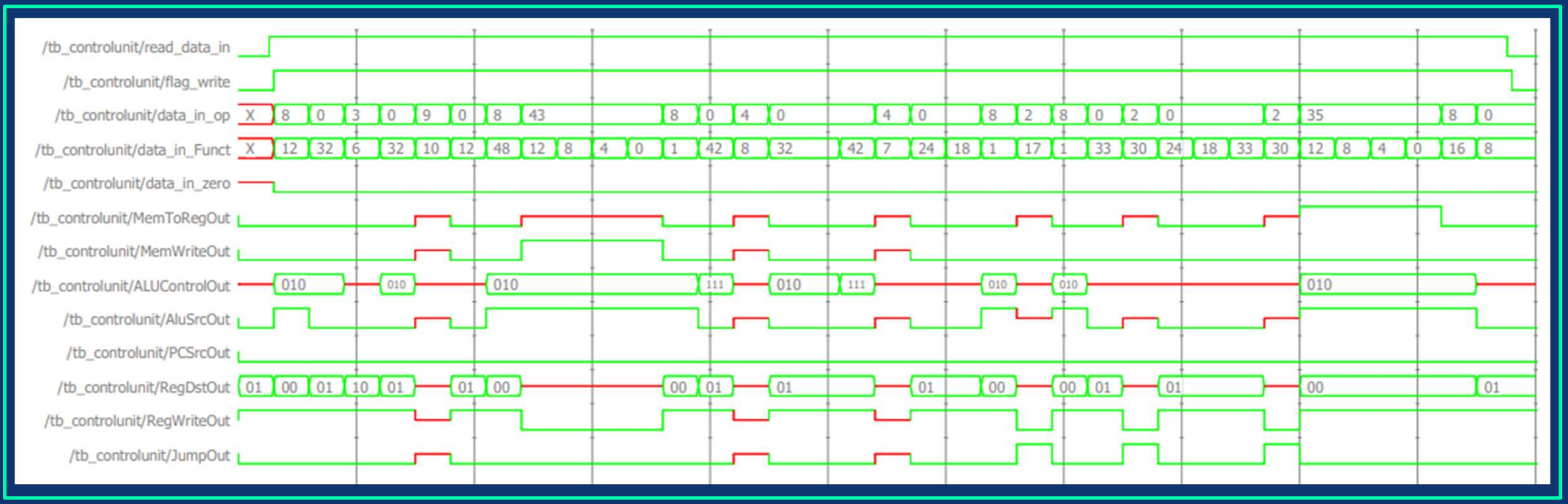
A *Control Unit* é a unidade em que o *Main Decoder* e o *ALU Decoder* fazem conexão entre si.

Sua função é gerar combinacionalmente os sinais para o controle dos componentes do *Data Path*. Dessa forma, a unidade de controle deve ter conhecimento da Instrução sendo realizada para estabelecimento dos sinais de saída.

# CONTROL UNIT



# CONTROL UNIT



# MIPS

MIPS > MIPS.vhd

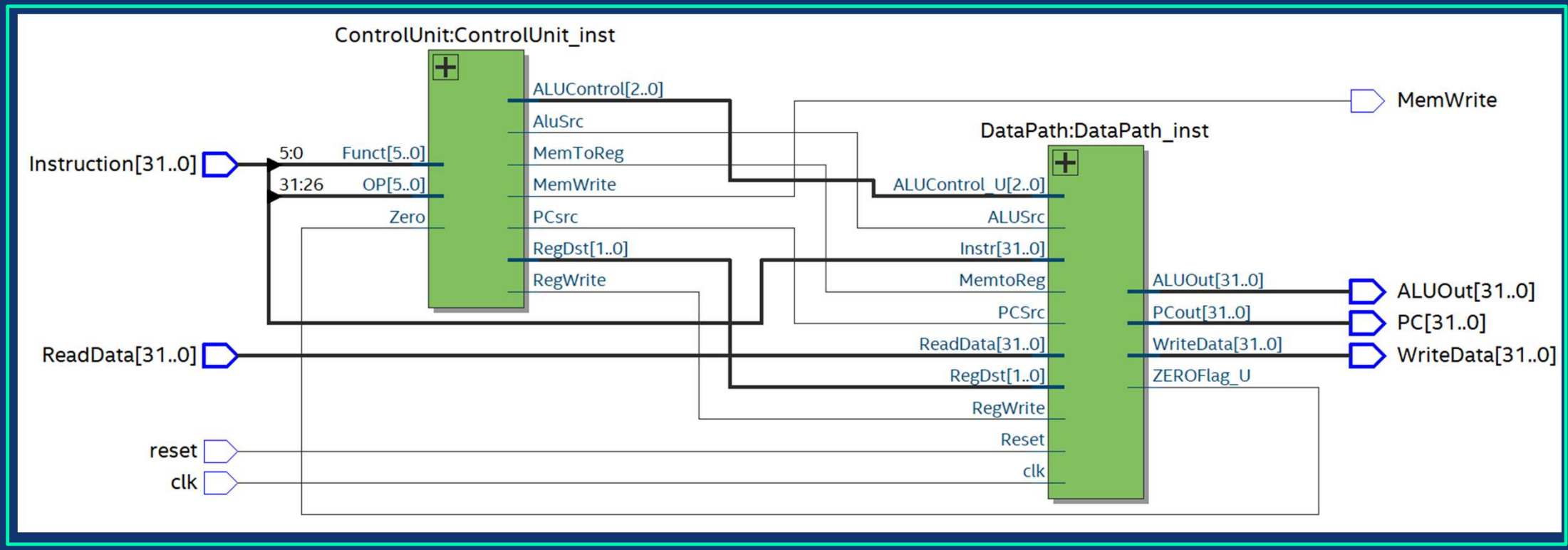
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity MIPS is
port(
    clk, reset   : in std_logic;
    Instruction : in std_logic_vector(31 downto 0);
    ReadData    : in std_logic_vector(31 downto 0);
    ALUOut      : out std_logic_vector(31 downto 0);
    PC          : out std_logic_vector(31 downto 0);
    WriteData   : out std_logic_vector(31 downto 0);
    MemWrite    : out std_logic
);
end MIPS;
```

Essa unidade descreve a conexão entre a Unidade de Controle e o Caminho de Dados e sua comunicação entre a Memória de Dados e Memória de Instruções.

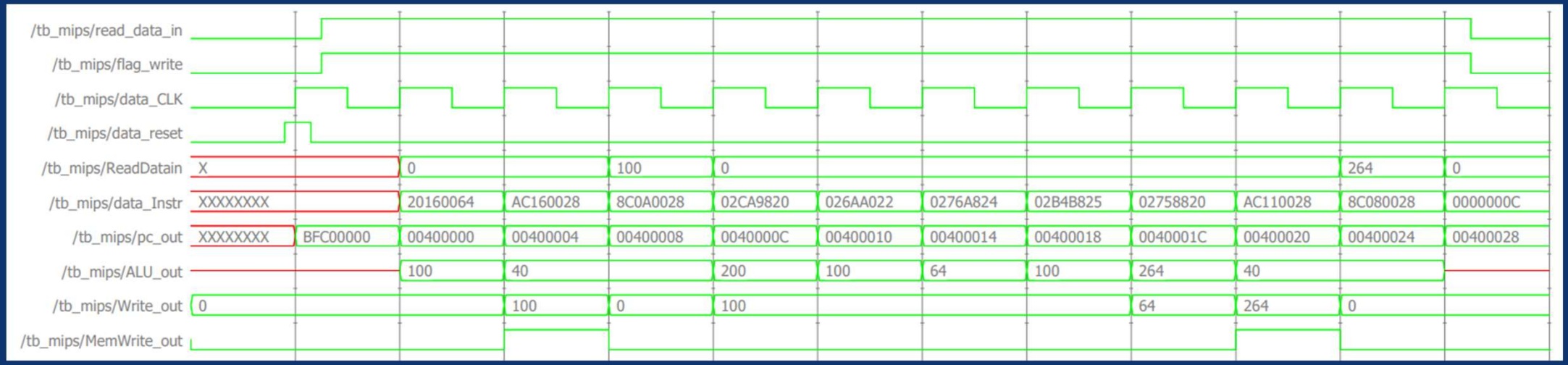
O processador MIPS é descrito separadamente da Memória de Dados e da Instrução de Memória.

# MIPS


ControlUnit:ControlUnit_inst
ALUControl[2..0]
AluSrc
MemToReg
MemWrite
PCsrc
RegDst[1..0]
RegWrite
Funct[5..0]
OP[5..0]
Zero
5:0
31:26
Instruction[31..0]
MemWrite
DataPath:DataPath_inst
ALUControl_U[2..0]
ALUSrc
Instr[31..0]
MemtoReg
PCSrc
ReadData[31..0]
RegDst[1..0]
RegWrite
Reset
clk
ALUOut[31..0]
PCout[31..0]
WriteData[31..0]
ZEROFlag_U
ReadData[31..0]
reset
clk
ALUOut[31..0]
PC[31..0]
WriteData[31..0]

0x01D

# MIPS




| Address | Code | Basic | Source |
|---|---|---|---|
| 0x00400000 | 0x20160064 | addi $22,$0,100 | 1 addi $s6, $0, 100 |
| 0x00400004 | 0xac160028 | sw $22,40($0) | 2 sw $s6, 40($0) |
| 0x00400008 | 0x8c0a0028 | lw $10,40($0) | 3 lw $t2, 40($0) |
| 0x0040000c | 0x02ca9820 | add $19,$22,$10 | 4 add $s3, $s6, $t2 |
| 0x00400010 | 0x026aa022 | sub $20,$19,$10 | 5 sub $s4, $s3, $t2 |
| 0x00400014 | 0x0276a824 | and $21,$19,$22 | 6 and $s5, $s3, $s6 |
| 0x00400018 | 0x02b4b825 | or $23,$21,$20 | 7 or $s7, $s5, $s4 |
| 0x0040001c | 0x02758820 | add $17,$19,$21 | 8 add $s1, $s3, $s5 |
| 0x00400020 | 0xac110028 | sw $17,40($0) | 9 sw $s1, 40($0) |
| 0x00400024 | 0x8c080028 | lw $8,40($0) | 10 lw $t0, 40($0) |
| 0x00400028 | 0x0000000c | syscall | 11 syscall |

# MIPS PROCESSOR

MIPS Processor > MipsProcessor.vhd

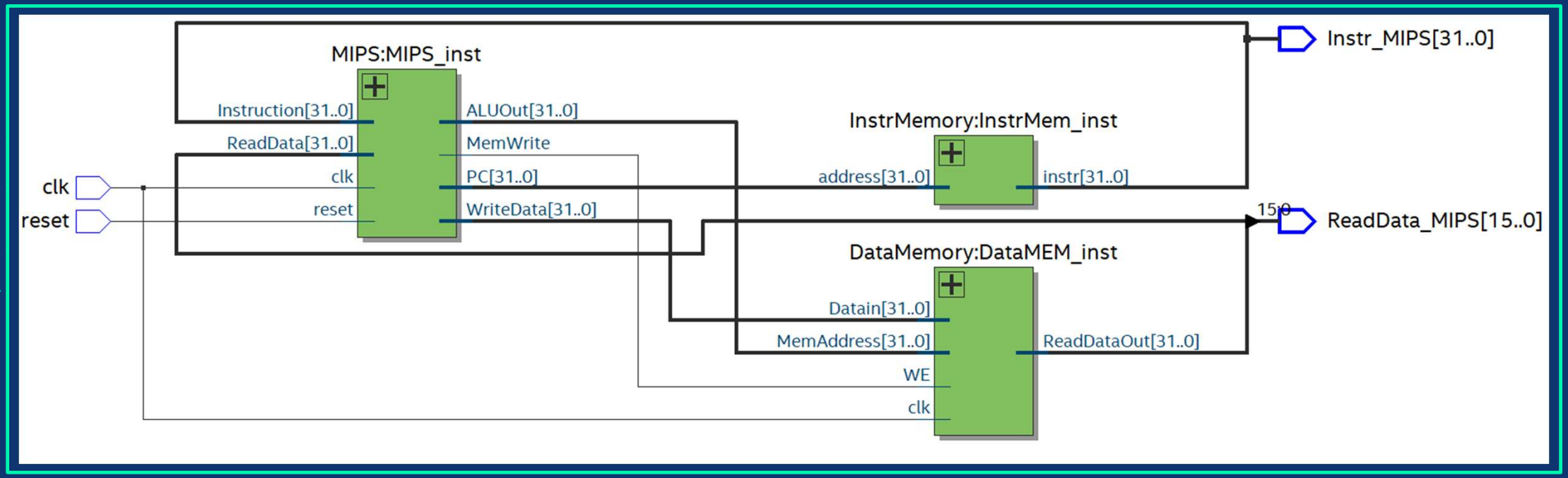
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity MipsProcessor is
    port(
        clk           : in std_logic;
        reset         : in std_logic;
        Instr_MIPS    : out std_logic_vector(31 downto 0);
        ReadData_MIPS : out std_logic_vector(15 downto 0)
    );
    end entity;
```

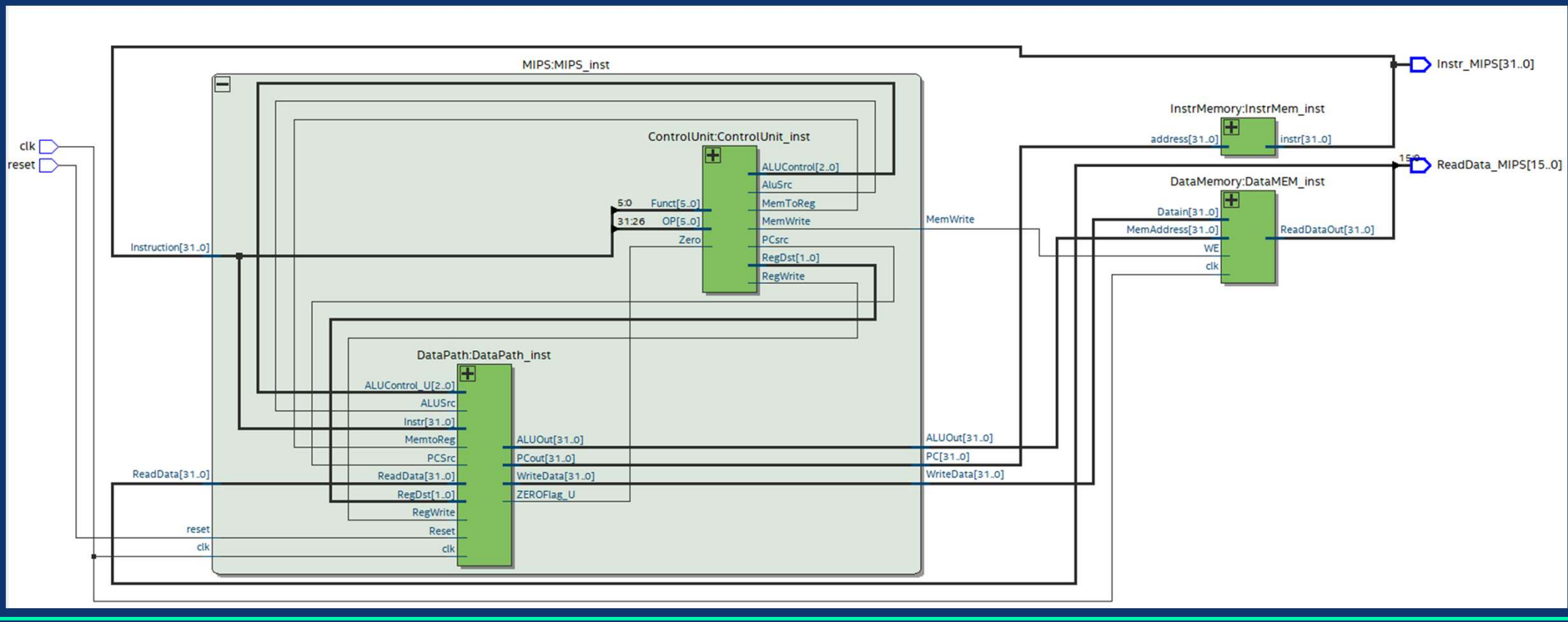
Essa unidade descreve a conexão entre a Unidade de Controle e o Caminho de Dados e sua comunicação entre a Memória de Dados e Memória de Instruções.

O processador MIPS é descrito separadamente da Memória de Dados e da Instrução de Memória.

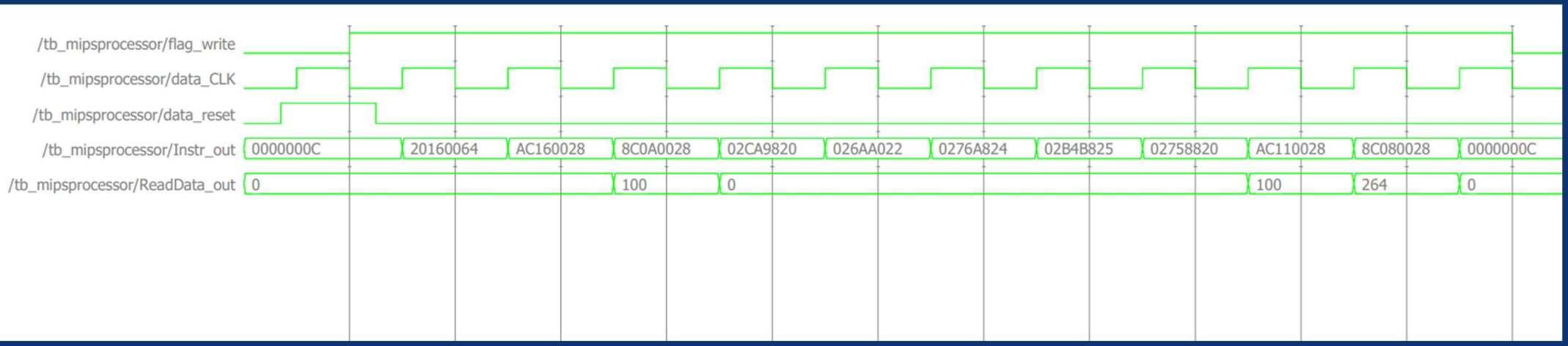
# MIPS PROCESSOR



# MIPS PROCESSOR



# MIPS PROCESSOR




| Address | Code | Basic | Source |
|---|---|---|---|
| 0x00400000 | 0x20160064 | addi $22,$0,100 | 1 addi $s6, $0, 100 |
| 0x00400004 | 0xac160028 | sw $22,40($0) | 2 sw $s6, 40($0) |
| 0x00400008 | 0x8c0a0028 | lw $10,40($0) | 3 lw $t2, 40($0) |
| 0x0040000c | 0x02ca9820 | add $19,$22,$10 | 4 add $s3, $s6, $t2 |
| 0x00400010 | 0x026aa022 | sub $20,$19,$10 | 5 sub $s4, $s3, $t2 |
| 0x00400014 | 0x0276a824 | and $21,$19,$22 | 6 and $s5, $s3, $s6 |
| 0x00400018 | 0x02b4b825 | or $23,$21,$20 | 7 or $s7, $s5, $s4 |
| 0x0040001c | 0x02758820 | add $17,$19,$21 | 8 add $s1, $s3, $s5 |
| 0x00400020 | 0xac110028 | sw $17,40($0) | 9 sw $s1, 40($0) |
| 0x00400024 | 0x8c080028 | lw $8,40($0) | 10 lw $t0, 40($0) |
| 0x00400028 | 0x0000000c | syscall | 11 syscall |

# TESTE NA PLACA

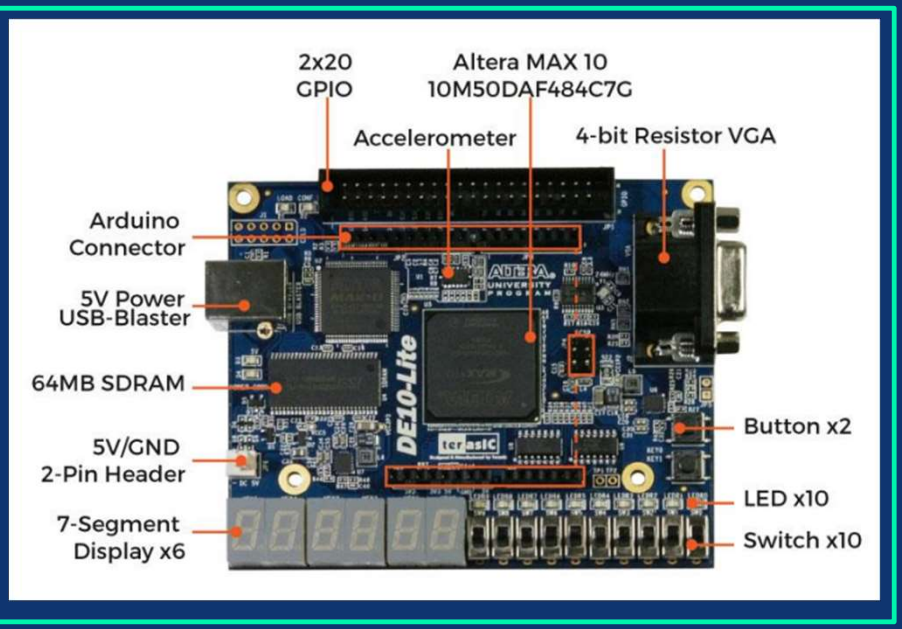
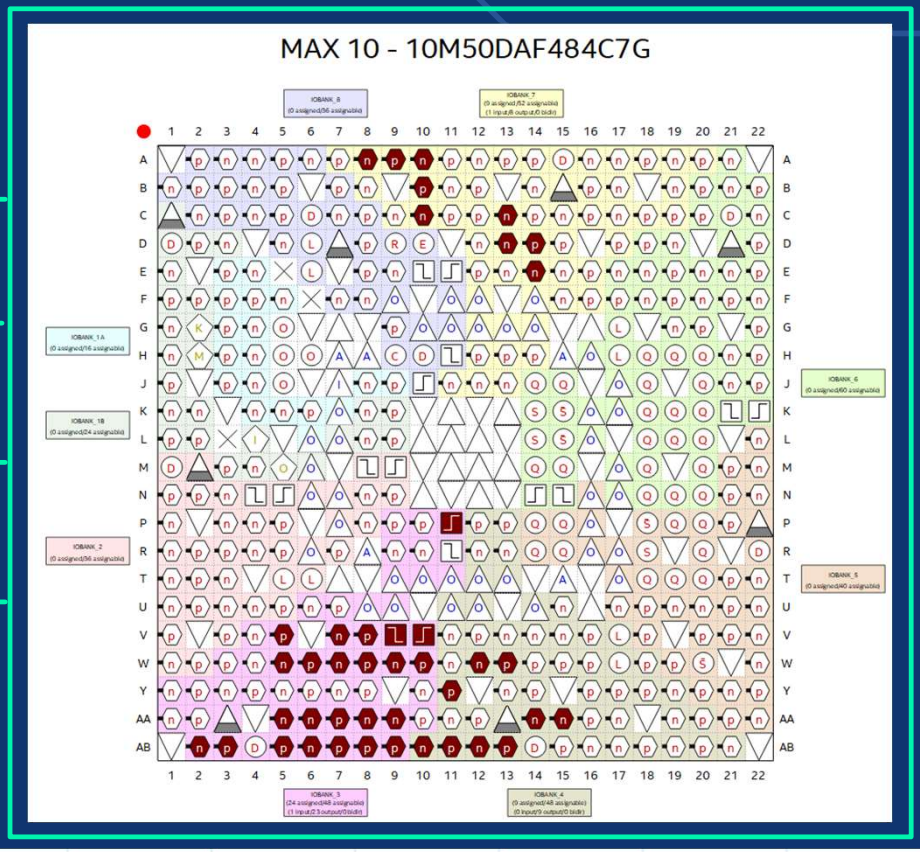



Table 3-5 Pin Assignment of LEDs

| Signal Name | FPGA Pin No. | Description | I/O Standard |
|---|---|---|---|
| LEDR0 | PIN_A8 | LED [0] | 3.3-V LVTTL |
| LEDR1 | PIN_A9 | LED [1] | 3.3-V LVTTL |
| LEDR2 | PIN_A10 | LED [2] | 3.3-V LVTTL |
| LEDR3 | PIN_B10 | LED [3] | 3.3-V LVTTL |
| LEDR4 | PIN_D13 | LED [4] | 3.3-V LVTTL |
| LEDR5 | PIN_C13 | LED [5] | 3.3-V LVTTL |
| LEDR6 | PIN_E14 | LED [6] | 3.3-V LVTTL |
| LEDR7 | PIN_D14 | LED [7] | 3.3-V LVTTL |
| LEDR8 | PIN_A11 | LED [8] | 3.3-V LVTTL |
| LEDR9 | PIN_B11 | LED [9] | 3.3-V LVTTL |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| out ReadData_MIPS[7] | Output | PIN_D14 | 7 | B7_N0 | PIN_D14 | 2.5 V | | 12mA (default) | 2 (default) |
| out ReadData_MIPS[6] | Output | PIN_E14 | 7 | B7_N0 | PIN_E14 | 2.5 V | | 12mA (default) | 2 (default) |
| out ReadData_MIPS[5] | Output | PIN_C13 | 7 | B7_N0 | PIN_C13 | 2.5 V | | 12mA (default) | 2 (default) |
| out ReadData_MIPS[4] | Output | PIN_D13 | 7 | B7_N0 | PIN_D13 | 2.5 V | | 12mA (default) | 2 (default) |
| out ReadData_MIPS[3] | Output | PIN_B10 | 7 | B7_N0 | PIN_B10 | 2.5 V | | 12mA (default) | 2 (default) |
| out ReadData_MIPS[2] | Output | PIN_A10 | 7 | B7_N0 | PIN_A10 | 2.5 V | | 12mA (default) | 2 (default) |
| out ReadData_MIPS[1] | Output | PIN_A9 | 7 | B7_N0 | PIN_A9 | 2.5 V | | 12mA (default) | 2 (default) |
| out ReadData_MIPS[0] | Output | PIN_A8 | 7 | B7_N0 | PIN_A8 | 2.5 V | | 12mA (default) | 2 (default) |
| in reset | Input | PIN_C10 | 7 | B7_N0 | PIN_C10 | 2.5 V | | 12mA (default) | |




| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| in clk | Input | PIN_P11 | 3 | B3_N0 | PIN_P11 | 2.5 V | | 12mA (default) | |
| out Instr_MIPS[31] | Output | PIN_V10 | 3 | B3_N0 | PIN_V10 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[30] | Output | PIN_V9 | 3 | B3_N0 | PIN_V9 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[29] | Output | PIN_V8 | 3 | B3_N0 | PIN_V8 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[28] | Output | PIN_V7 | 3 | B3_N0 | PIN_V7 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[27] | Output | PIN_W6 | 3 | B3_N0 | PIN_W6 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[26] | Output | PIN_W5 | 3 | B3_N0 | PIN_W5 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[25] | Output | PIN_AA14 | 4 | B4_N0 | PIN_AA14 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[24] | Output | PIN_W12 | 4 | B4_N0 | PIN_W12 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[23] | Output | PIN_AB12 | 4 | B4_N0 | PIN_AB12 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[22] | Output | PIN_AB11 | 4 | B4_N0 | PIN_AB11 | 2.5 V | | 12mA (default) | 2 (default) |
| out Instr_MIPS[21] | Output | PIN_AB10 | 4 | B4_N0 | PIN_AB10 | 2.5 V | | 12mA (default) | 2 (default) |

# ANÁLISE DE CLOCK

- Fmax: Frequência máxima de clock que o design suporta sem violar a configuração interna e tempo de acesso;
- Fmax pode ser maior ou menor, dependendo do esforço do *Fitter* no processo de roteamento;

*Auto-fit:* quão logo encontra uma configuração com requisitos mínimos;
*Standard fit:* busca a melhor configuração para o sistema;

*Auto-fit* com requerimento de clock de 50 MHz:

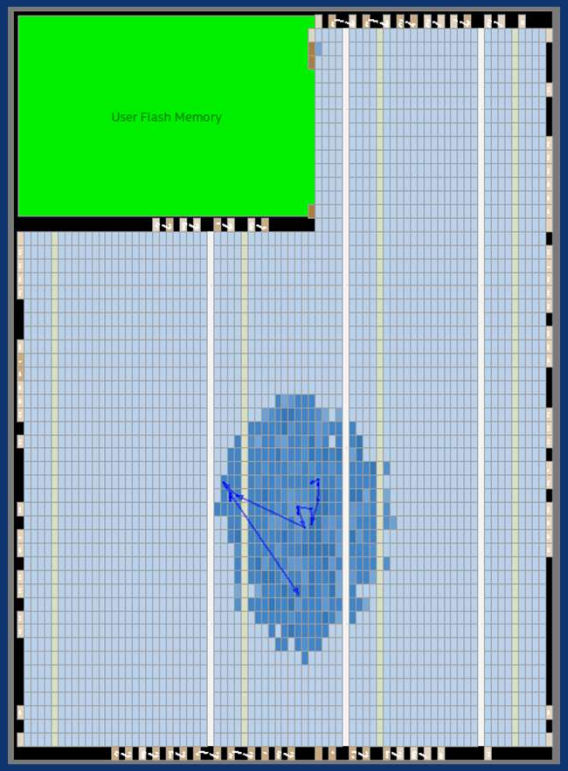
| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 49.92 MHz | 49.92 MHz | clcok | |

*Standard-Fit* com requerimento de clock de 50 MHz:

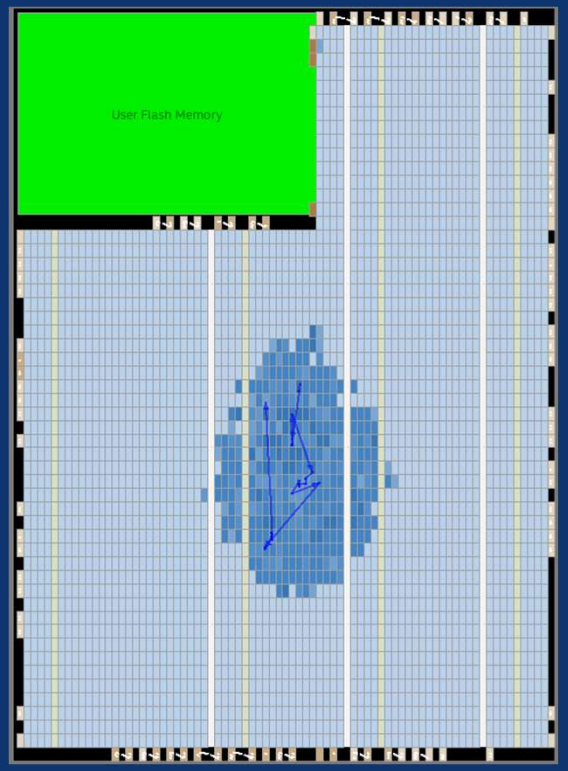
| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 51.63 MHz | 51.63 MHz | clcok | |

# ANÁLISE DE CLOCK

Auto-fit com requerimento de clock de 50 MHz:

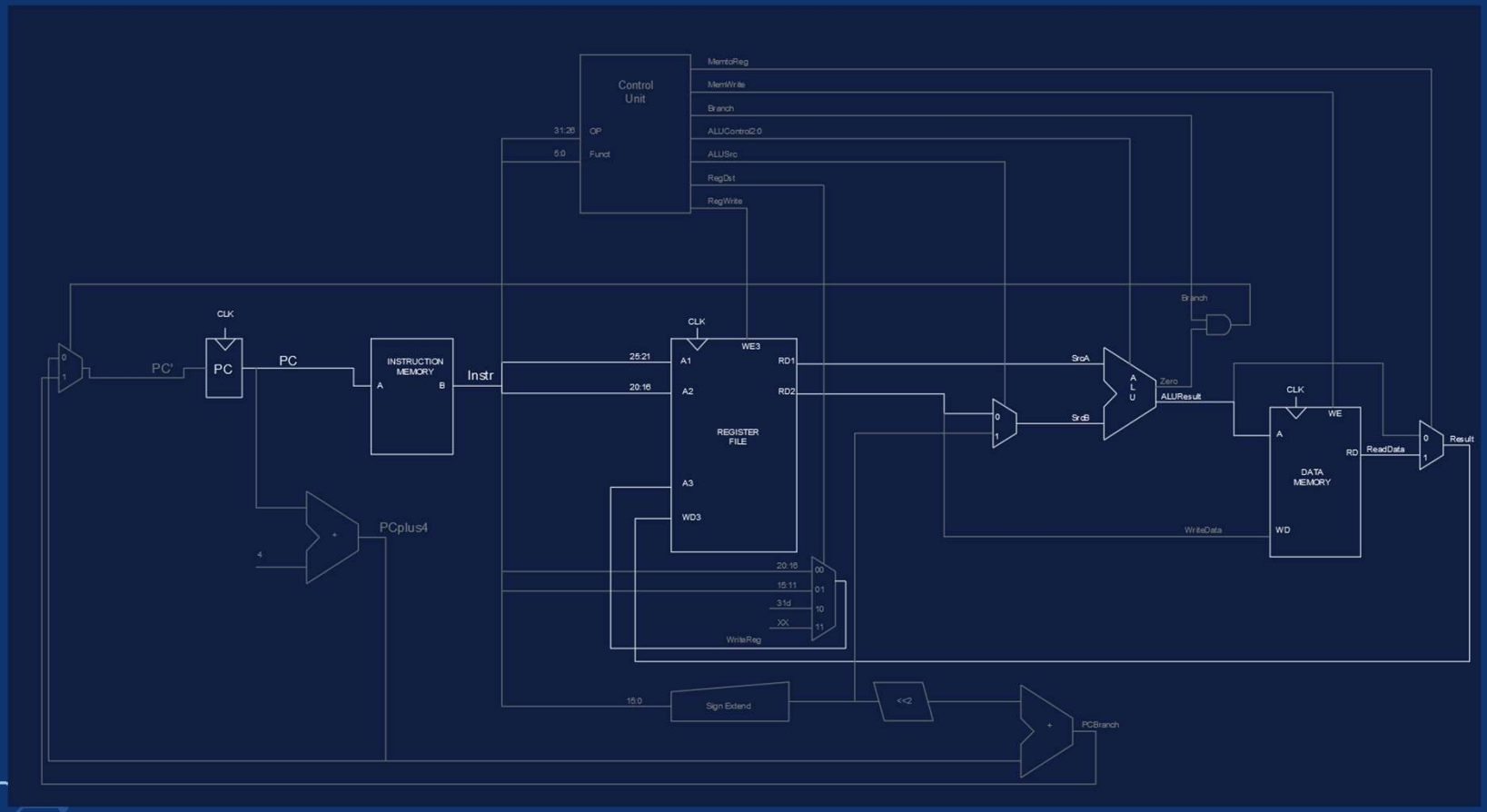



Standard-Fit com requerimento de clock de 50 MHz:



# ANÁLISE DE CLOCK

Caminho crítico: caminho com os maiores atrasos do clock;



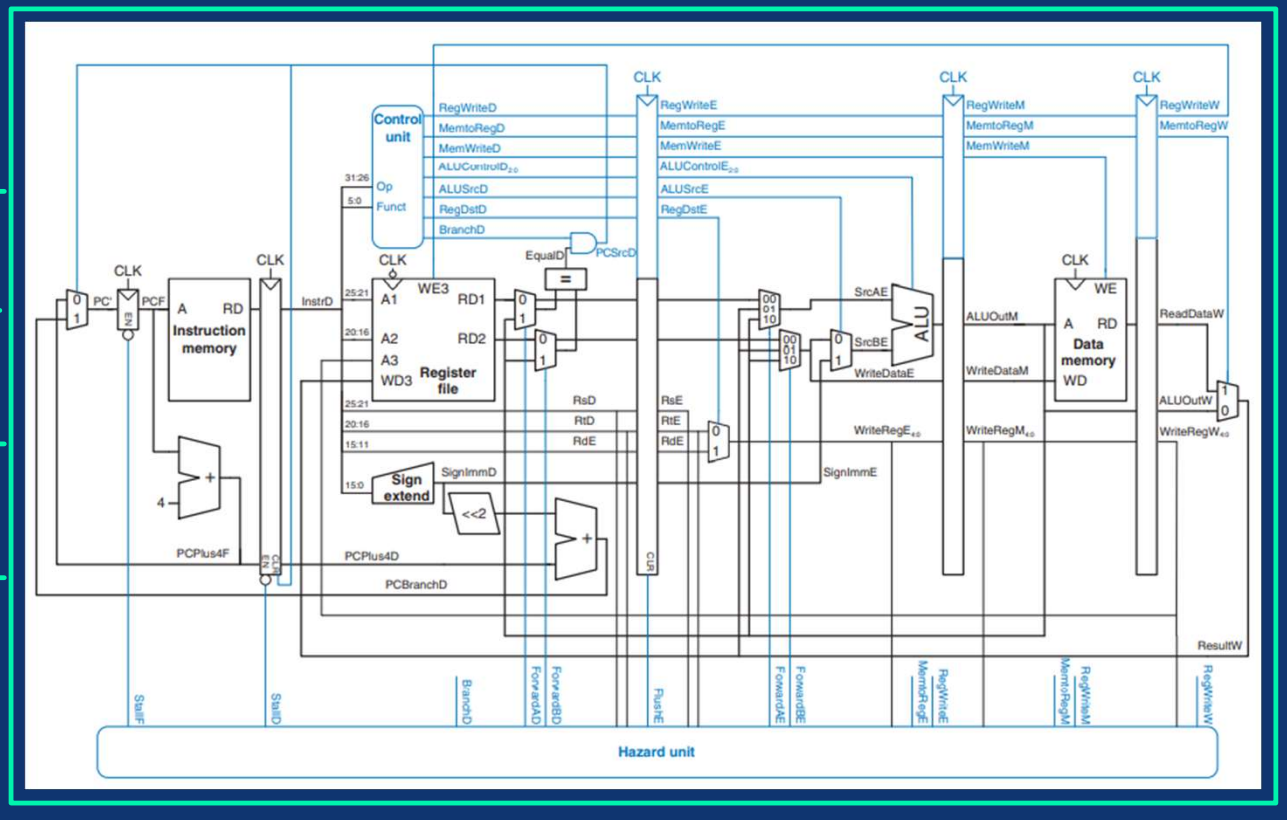
0x027


# CONCLUSÃO

Melhorias e pipeline

# MELHORIAS

- Algumas instruções possuem o hardware implementado para sua execução, mas não a lógica:
- Instruções tipo Branch, Jump e Set Less Than;
- Implementação utilizando a prórpria SDRAM da placa DE-10;
- Pipeline:



# DÚVIDAS?

victorcesarts/Final (github.com)