

UNIVERSIDADE FEDERAL DE SANTA CATARINA

JOSÉ LUIZ DE SOUZA

**IMPLEMENTAÇÃO E SIMULAÇÃO DO PROCESSADOR MIPS MULTICICLO COM VHDL**

Florianópolis

2019

[**INTRODUÇÃO**](#_fn7b80wg8xcl) **3**

[**INSTRUCTION SET**](#_4az8rwmvg95i) **5**

[**MÓDULOS VHDL**](#_uuk20qmyct7o) **6**

[And\_Or](#_pyb3ul8aucye) 6

[bancoRegs](#_ioqzq3sv5rcm) 6

[bitsReg](#_6s8jp48zop9y) 7

[Controle](#_cgh9sy35tmsr) 7

[Datapath](#_4uj6oxbo3fsk) 7

[DatapathDBG](#_6c809wfj8j1k) 7

[Deslocador1 e Deslocador2](#_b3tdiampceeh) 7

[Memória](#_5u0fhd430hhw) 7

[MIPS](#_qyw3y2tjw89t) 7

[Mux2x1 e Mux4x1](#_qv9oshnwoamk) 8

[PC](#_npzfqn5k4o87) 8

[regInstrução](#_cbjn9bmlcnni) 8

[Signal\_Extender](#_avywdgsnitee) 8

[Sum\_Sub](#_4ysvon4cql7r) 8

[ULA, ULA\_Control e ULA\_Wrapper](#_j6ypfpg4z34) 8

[**SIMULAÇÃO**](#_9ghdo08ahbu5) **8**

[INSTRUÇÃO DO TIPO R](#_m2wtgskhz2ba) 9

[SOMA](#_kuatmbwbicym) 9

[SUBTRAÇÃO](#_v8dczlam721q) 10

[AND](#_4nrbphu1lcrm) 10

[OR](#_694nr3rh9al) 11

[SET LESS THAN](#_sd568mr59t18) 11

[INSTRUÇÃO DO TIPO BEQ](#_oebzq0w1aixx) 12

[$s1 =/= $s2](#_gl91ui3pi9ib) 12

[$s1 == $s2](#_frxt3rijoq0x) 13

[INSTRUÇÃO DO TIPO JUMP](#_q1ozpzp9n8tx) 13

[INSTRUÇÃO DO TIPO LW E SW](#_xgsilmvxnxa2) 14

[STORE WORD](#_culwzyfv3ep1) 14

[**DADOS DE UTILIZAÇÃO DA PLACA**](#_23kha8e5qqg9) **14**

[COMPILATION REPORT](#_6b9j346t80xh) 15

[PROPAGATION DELAY](#_mq7u94lisyc0) 15

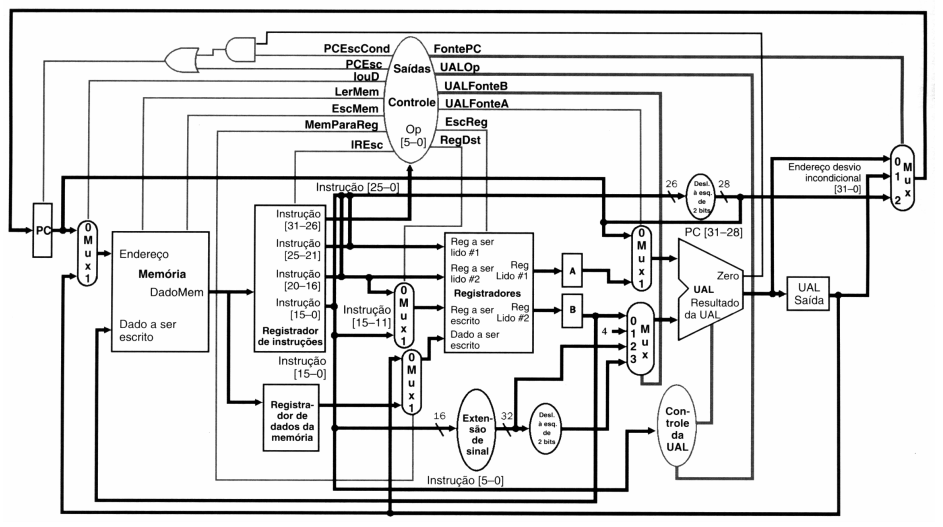
[MINIMUM CLOCK TO OUTPUT TIMES](#_ers3x48fbte) 16

[**CONCLUSÃO**](#_7qo9ao7f4700) **16**

# INTRODUÇÃO

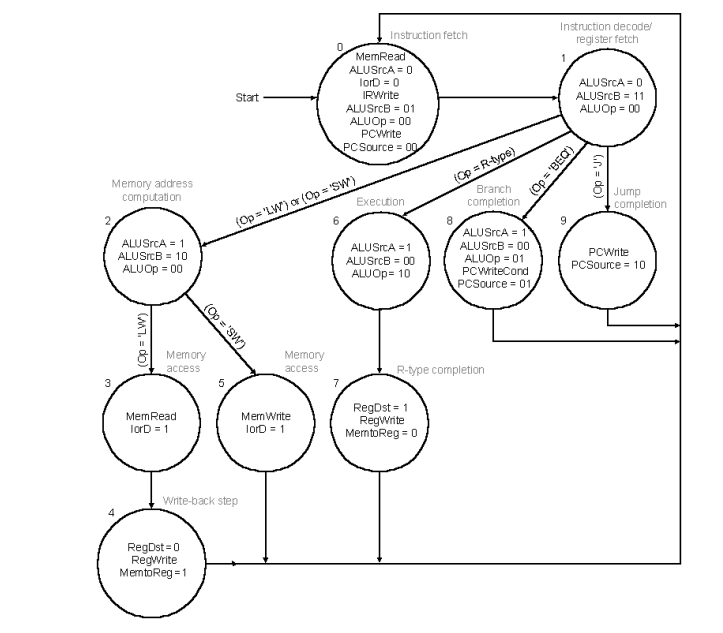
Esse relatório explica a simulação de um processador MIPS Multiciclo, cuja implementação foi realizada na linguagem VHDL.

O modelo implementado é uma versão mais simplificada do MIPS, sem conter algumas instruções e seu modelo se encontra na imagem abaixo.



Fonte : Organização e projeto de computadores - David A. Patterson

Sua implementação foi realizada no modelo RTL, contendo uma Máquina de Estados Finitos no controlador e o datapath, que contém todos os submódulos do processador. O modelo da FSM está descrita na imagem abaixo.



Fonte : Organização e projeto de computadores - David A. Patterson

A simulação foi realizada utilizando o ModelSim Altera, com o projeto compilado e configurado para rodar na placa Cyclone II, modelo EP2C35F672C6.

# INSTRUCTION SET

As operações realizadas pelo MIPS estão descritas na tabela abaixo.

|  |  |  |
| --- | --- | --- |
| Instrução do tipo R | Realiza Operações na ULA | $s1 = $s2 op $s3 |
| Instrução SW e LW | Busca e Salva dados na memória | $s1 = Mem[$s2 + desl]  Mem[$s2 + desl] = $s1 |
| Instrução BEQ | Desvio condicional | if $s1 == $s1 -> desvio |
| Instrução Jump | Desvio Incondicional | desvio |

Dessas quais a Instrução do tipo R realiza as operações de soma, subtração, “e” lógico, “ou” lógico e Set less than.

Outras instruções não estão incluídas na implementação, como realizar operações entre um registrador e uma constante, ou uma multiplicação/divisão.

# MÓDULOS VHDL

A implementação em VHDL foi realizada desenvolvendo cada parte do MIPS separadamente e depois indexada em um arquivo topo.

Inicialmente o projeto foi planejado para possuir um arquivo principal na qual tivesse um port map para o datapath, que incluiria todos os submódulos, exceto o controle, e o controle. Isso se manteve no projeto, porém os testes e simulações feitas foram em um único arquivo datapath que possui todos os ports maps do processador.

Os módulos incluídos no projeto estão descritos a seguir.

## And\_Or

Módulo que realiza a operação de AND e OR lógico entre dois valores. Utilizado pela ULA.

## bancoRegs

Módulo que realiza a função de banco de registradores do MIPS.

## bitsReg

Módulo que representa um registrador. Utilizado como registrador da ULASaida, Registrador de dados, registrador A e registrador B (Referentes a saida do banco de registradores).

## Controle

Possui a Máquina de Estados Finitos que controla o processador.

## Datapath

Módulo que faz a ligação entre todas as partes do MIPS. Esse módulo é utilizado em conjunto com o Controle em um arquivo topo superior.

## DatapathDBG

Módulo que possui todos as ligações entre módulos do projeto, sendo esse módulo o arquivo topo compilado para simular o processador. O controle está incluído.

## Deslocador1 e Deslocador2

Responsáveis pelo deslocamento de bits utilizados pelo MIPS.

## Memória

Módulo de memoria RAM principal, na qual lê e escreve instruções e dados.

## MIPS

Arquivo TOPO. Possui port map para o datapath e controle.

## Mux2x1 e Mux4x1

Módulo que representa um multiplexador.

## PC

Registrador que guarda o endereço da próxima instrução.

## regInstrução

Registrador que guarda a instrução que está sendo executada.

## Signal\_Extender

Módulo responsável por estender o sinal para 32 bits.

## Sum\_Sub

Módulo responsável por realizar operações de Soma e Subtração entre dois números.

## ULA, ULA\_Control e ULA\_Wrapper

Módulos que juntos foram a ULA do processador. Realiza todas as operações aritméticas e lógicas do processador.

# SIMULAÇÃO

Será apresentado os resultados da simulação do MIPS.

A Simulação foi realizada forçando os valores da saída do registrador PC, saída do registrador de Instrução e das saídas do banco de registradores.

Isso possui uma limitação óbvia, onde não há como observar totalmente o funcionamento do processador, porém essa foi a forma encontrada pelo autor de realizar os testes.

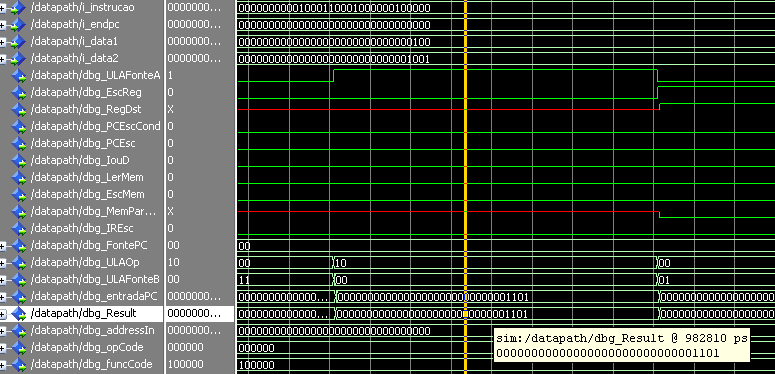
Com essa limitação, torna-se impossível observar resultados de simulação da Instrução Load Word. A Instrução Store Word é possível observar os valores de entrada na memória e dos sinais do controle no momento, podendo supor seu funcionamento correto.

## INSTRUÇÃO DO TIPO R

A simulação do tipo R inclui as operações de SOMA, SUBTRAÇÃO, AND, OR e SET LESS THAN.

### SOMA

**add $s1 $s2 $s3**



**force /i\_instrucao 00000000001000110001000000100000 0 ns**

**force /i\_endpc 00000000000000000000000000000000 0 ns**

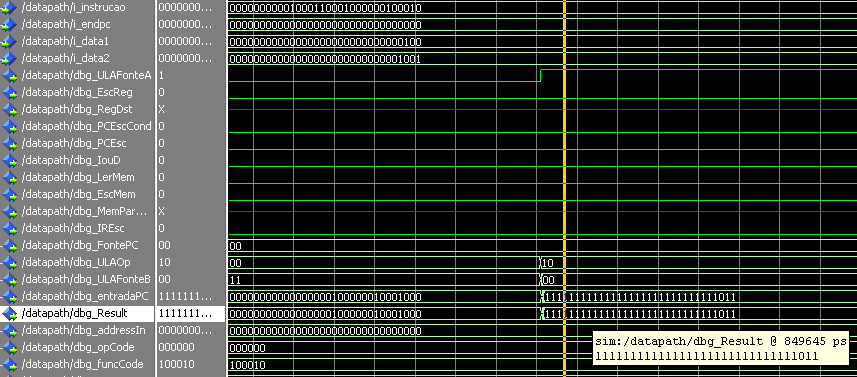
**force /i\_data1 00000000000000000000000000000100 0 ns**

**force /i\_data2 00000000000000000000000000001001 0 ns**

Ao realizar a soma entre i\_data1 e i\_data2 temos a saída como “00000000000000000000000000001101”, que é o resultado esperado.

### SUBTRAÇÃO

**sub $s1 $s2 $s3**

****

**force /i\_instrucao 00000000001000110001000000100010 0 ns**

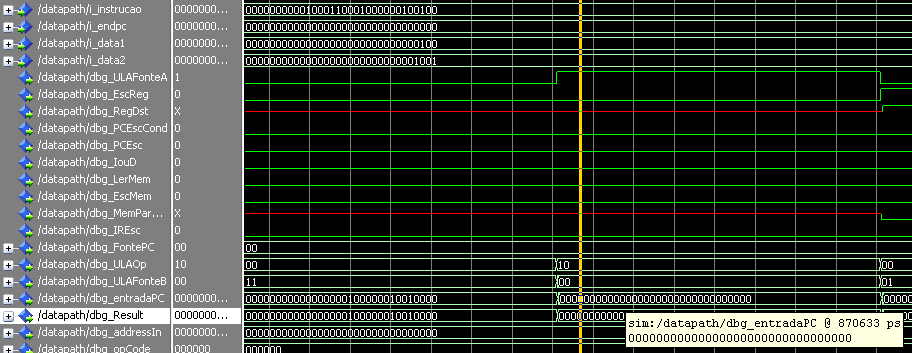
**force /i\_endpc 00000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

**force /i\_data2 00000000000000000000000000001001 0 ns**

Ao realizar a subtração entre i\_data1 e i\_data2 temos a saída como “11111111111111111111111111111011”, que é o resultado esperado.

### AND



**force /i\_instrucao 00000000001000110001000000100100 0 ns**

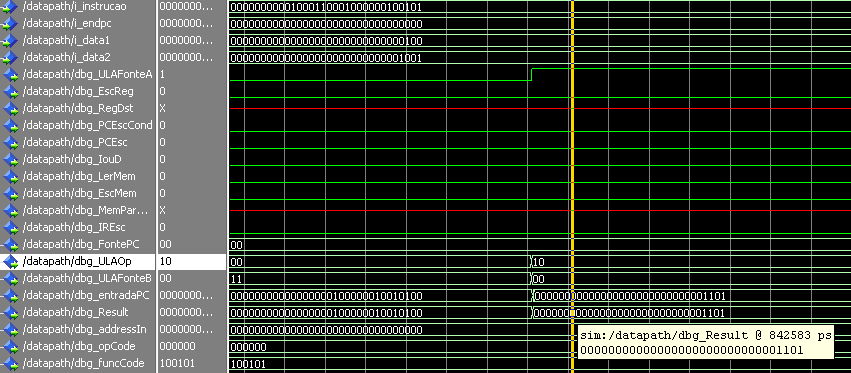
**force /i\_endpc 00000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

**force /i\_data2 00000000000000000000000000001001 0 ns**

Ao realizar a operação AND entre i\_data1 e i\_data2 temos a saída como “00000000000000000000000000000000”, que é o resultado esperado.

### OR



**force /i\_instrucao 00000000001000110001000000100101 0 ns**

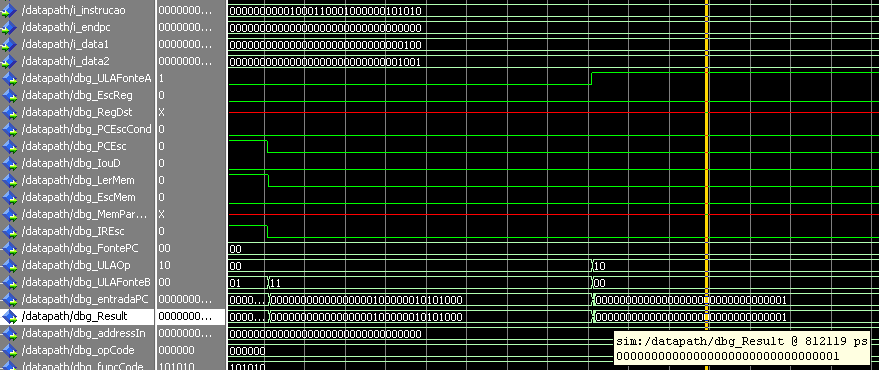
**force /i\_endpc 00000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

**force /i\_data2 00000000000000000000000000001001 0 ns**

Ao realizar a operação OR entre i\_data1 e i\_data2 temos a saída como “00000000000000000000000000001101”, que é o resultado esperado.

### SET LESS THAN



**force /i\_instrucao 00000000001000110001000000101010 0 ns**

**force /i\_endpc 00000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

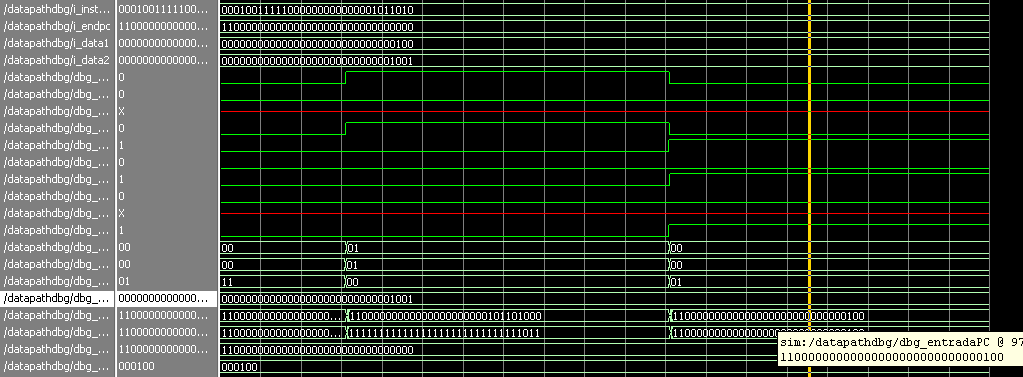
**force /i\_data2 00000000000000000000000000001001 0 ns**

Ao realizar a operação SET LESS THAN entre i\_data1 e i\_data2 temos a saída como “00000000000000000000000000000001”, que é o resultado esperado.

## INSTRUÇÃO DO TIPO BEQ

Instrução do tipo desvio condicional, onde o PC irá desviar para um endereço na instrução caso os registradores rs e rt forem iguais.

### $s1 =/= $s2

****

**force /i\_instrucao 00010011111000000000000001011010 0 ns**

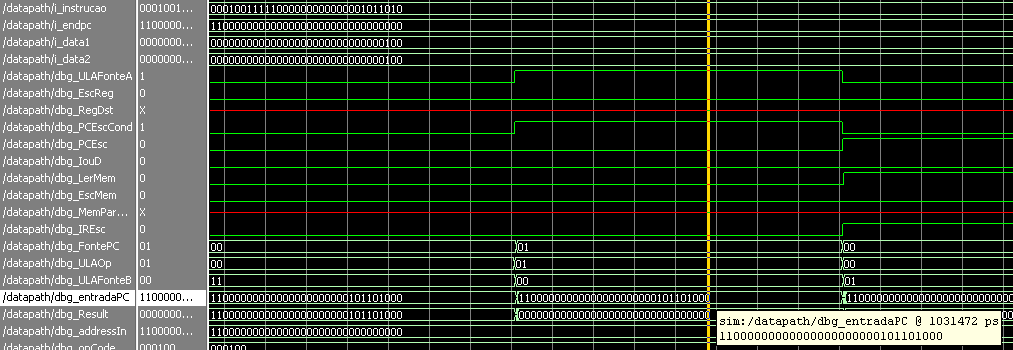
**force /i\_endpc 11000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

**force /i\_data2 00000000000000000000000000001001 0 ns**

Considerando o valor de i\_data1 diferente de i\_data2, o valor do PC apenas é incrementado em 4, pulando para a próxima instrução normalmente.

### $s1 == $s2



**force /i\_instrucao 00010011111000000000000001011010 0 ns**

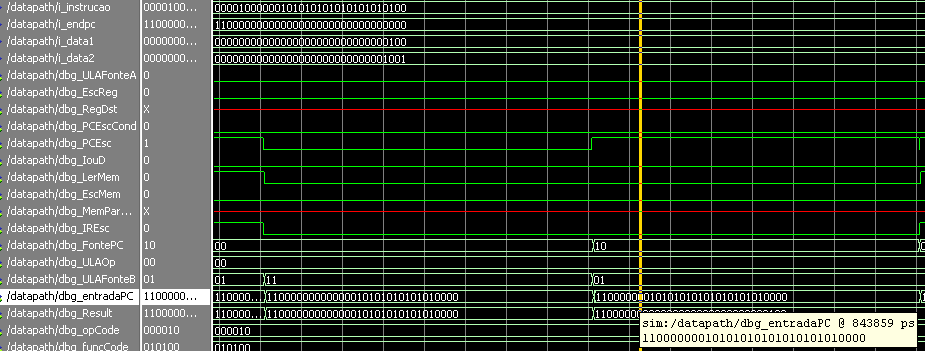
**force /i\_endpc 11000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

**force /i\_data2 00000000000000000000000000000100 0 ns**

Considerando o valor de i\_data1 igual ao de i\_data2, o valor do PC é desviado.

## INSTRUÇÃO DO TIPO JUMP



**force /i\_instrucao 00001000000101010101010101010100 0 ns**

**force /i\_endpc 11000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

**force /i\_data2 00000000000000000000000000001001 0 ns**

O valor do endereço do PC é simplesmente desviado incondicionalmente.

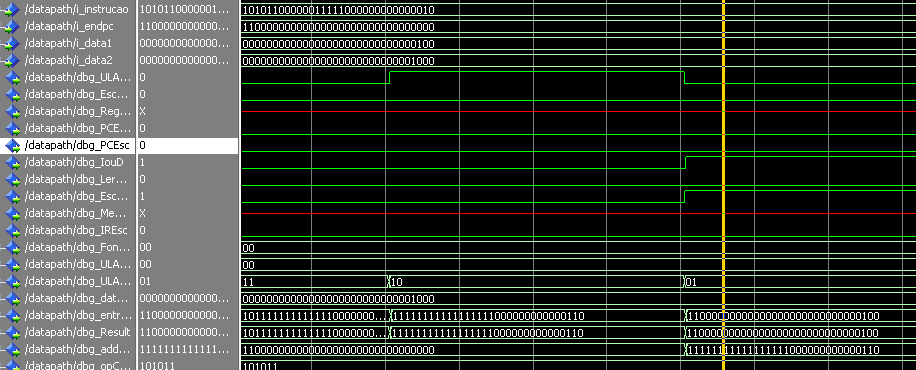
## INSTRUÇÃO DO TIPO LW E SW

Pelo modo que a simulação foi feita, as intruções do tipo SW e LW não são possiveis de serem simuladas.

Porém é possível obter as entradas da memória para a instrução do tipo Store Word, podendo implicar em seu funcionamento. Não é possível ver o dado sendo escrito, porém o dado correto e o endereço de escrita correto chega até os inputs da memória.

### STORE WORD

**Mem[$s2 + desl] = $s1**

****

**force /i\_instrucao 10101100000011111000000000000010 0 ns**

**force /i\_endpc 11000000000000000000000000000000 0 ns**

**force /i\_data1 00000000000000000000000000000100 0 ns**

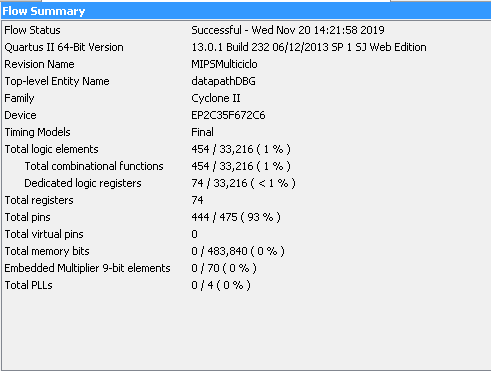
**force /i\_data2 00000000000000000000000000001000 0 ns**

Ao simular essa instrução, as entradas na memória são referentes ao dbg\_adressin que se encontra com o valor “11111111111111111000000000000110” e o valor de i\_data2, que será escrito no endereço de dbg\_adressin.

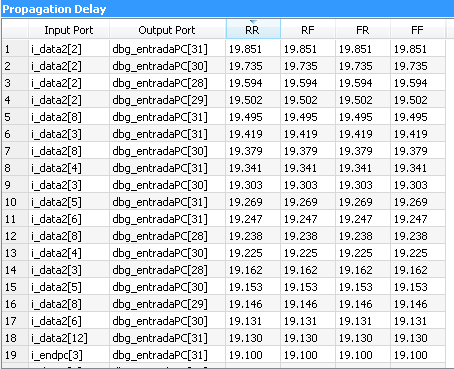
Perceba que o valor do sinal EscMem está em 1, implicando que irá escrever na memória naquele clock.

# DADOS DE UTILIZAÇÃO DA PLACA

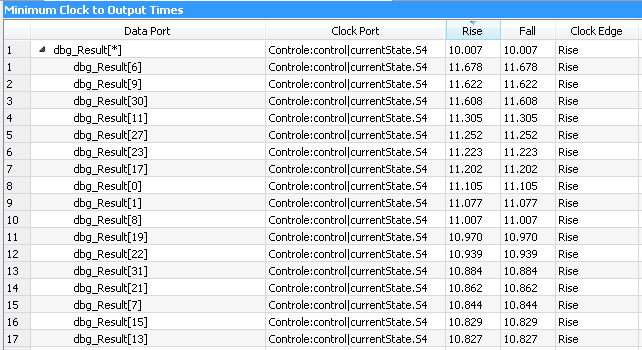
## COMPILATION REPORT



## PROPAGATION DELAY



## MINIMUM CLOCK TO OUTPUT TIMES



# CONCLUSÃO

Como o modo de simulação inclui forçar os valores de saída da memória, o processador não necessariamente tinha acesso a memória, podendo isso ter influenciado os valores de clock.

As quantidades de registradores utilizados também podem ter sido influenciados pela simulação.

Os circuitos combinacionais funcionam de acordo com a descrição do processador MIPS citado na introdução, com exceção da memória que não pode ser testada.

A utilização de algum método de incluir as instruções na memória antes de iniciar os testes permitiria a realização dos testes de forma funcional e totalmente correta, porém o mesmo não foi encontrado.