UNIVERSIDADE DE BRASÍLIA – UnB

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – CIC

PROJETO FINAL DA DISCIPLINA
ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Alexandre Aragão Souza Coelho – 13/0004880

Amanda Aline Figueiredo Carvalho – 15/0115997

Gabriel Rocha Fontenele – 15/0126760

Lucas Bonifácio Perez Nunes – 14/0056581

Brasília – DF

2017

1.INTRODUÇÃO

O trabalho consiste na implementação de um sistema em VHDL para simular o processador MIPS Uniciclo, onde todas as etapas do ciclo de instrução devem ocorrer em um único ciclo de clock. O sistema deve conter 32 registradores, cada um com 32 bits, que estarão em um banco de registradores. Também dá suporte às instruções LW, SW, ADD, ADDI, SUB, AND, ANDI, OR, ORI, NOR, XOR, XORI, SLT, SLTI, SLL, SRL, SRA, J, JR, JAL, BEQ, BNE. A palavra de processamento no sistema é um vetor de 32 bits, assim como as instruções com que trabalha o processador.

Sendo assim, foram implementados módulos específicos, bem como estabelecidos os sinais que o controle deve enviar ao datapath, de forma que satisfaçam as exigências para o correto funcionamento do datapath da CPU.

2. SOFTWARES UTILIZADOS

Para elaboração do MIPS Uniciclo, foram utilizados os softwares ModelSim-Altera, para análise das formas de onda das instruções e Quartus II, para codificação em VHDL dos testbenchs e do datapath.

3. IMPLEMENTAÇÃO DOS MÓDULOS

• MULTIPLEXADOR 2X1

A partir de um seletor, é selecionada uma das entradas e enviada à saída. Foi implementado um multiplexador com entradas de 5 bits e vários de 32 bits, cada um ativando operações específicas descritas no código VHDL.

• SOMADOR

Efetua a soma entre duas entradas e envia o resultado à saída.

• ULA

Realiza determinada operação com as duas entradas. Essa operação é definida pelo Controlador da ULA.

• BANCO DE REGISTRADORES

Contém 32 registradores, que serão utilizados durante a execução do programa. Foi implementado de forma algorítmica, usando 32 sinais de 32 bits.

• **EXTENSOR DE SINAL**

Transforma uma entrada de 16 bits em uma entrada de 32 bits. Dessa forma, concatena a entrada com 16 bits '0'. Dessa forma, conseguimos transformar valores imediatos (instrução tipo-I), em valores de 32 bits. Conseguimos também transformá-los em endereço de PC para instruções de jump, pois o valor de endereço usado nesta implementação nunca ultrapassa 7 bits.

Há ainda uma entrada para extensão de 5 bits para 32 bits. Essa entrada recebe o campo Shamt, e é usada para instruções de shift.

• **CONTROLADOR DA ULA**

Define a operação a ser executada através da ULA. Para isso, esse módulo recebe a saída ALUop como entrada e avalia o campo function para instruções tipo-R.

	ALUop		FUNCTION						OPERATION			
add	0	0	1	0	0	0	0	0	0	0	1	0
sub	0	0	1	0	0	0	1	0	0	0	1	1
and	0	0	1	0	0	1	0	0	0	0	0	0
or	0	0	1	0	0	1	0	1	0	0	0	1
nor	0	0	1	0	0	1	1	1	0	1	0	1
xor	0	0	1	0	0	1	1	0	0	1	1	0
Slt	0	0	1	0	1	0	1	0	0	1	0	0
Sll	0	0	0	0	0	0	0	0	0	1	1	1
Srl	0	0	0	0	0	0	1	0	1	0	0	0
Sra	0	0	0	0	0	0	1	1	1	0	0	1
Lw	1	0	1	0	0	0	1	1	0	0	1	0
Sw	1	0	1	0	1	0	1	1	0	0	1	0
addi	0	1	0	0	1	0	0	0	0	0	1	0
andi	0	1	0	0	1	1	0	0	0	0	0	0
Ori	0	1	0	0	1	1	0	1	0	0	0	1
xori	0	1	0	0	1	1	1	0	0	1	1	0
slti	0	1	0	0	1	0	1	0	0	1	0	0
beq	1	1	0	0	0	1	0	0	0	0	1	1
bne	1	1	0	0	0	1	0	1	0	0	1	1

Note que a partir da instrução LW, o Controle da ULA passa a conferir o campo Opcode das instruções. A tabela serve de base para as expressões booleanas usadas no código VHDL.

• **CONTROLADOR**

Faz atribuições de sinais de controle para cada comando a partir da instrução que recebe.

• **MEMÓRIA DE DADOS**

Local onde são armazenados dados que não estão em uso no momento. Foi implementado com um vetor de sinais de 32 bits. Com o endereço, transforma-se o valor em um inteiro e tem acesso àquela posição na memória.

• **CONTADOR DE PROGRAMA (PC)**

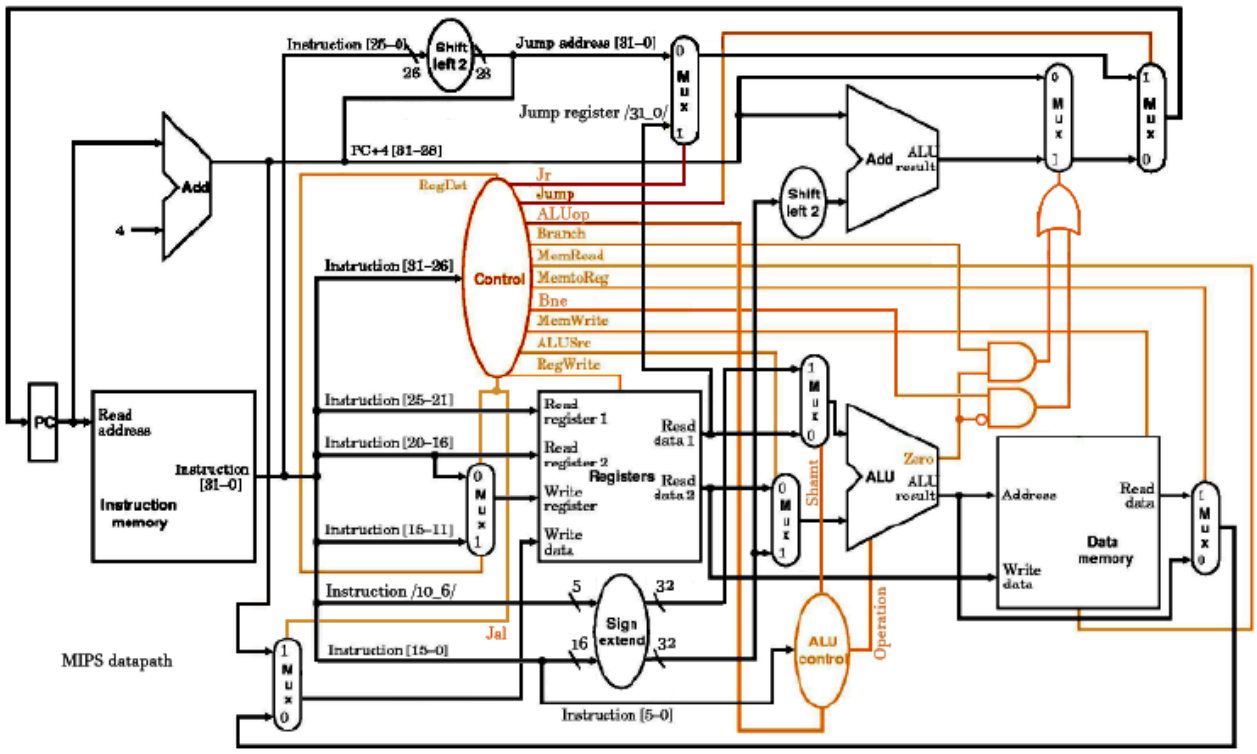
Registrador que contém o endereço da próxima instrução a ser executada.

• **DESLOCADOR DE BITS À ESQUERDA**

Desloca a informação dois bits para a esquerda concatenando com dois bits '0'.

4. DATAPATH DA CPU

A imagem a seguir mostra o datapath da CPU, ou seja, o caminho que os dados percorrem para se obter o resultado correto das operações ordenadas.



4.1. SINAIS DE CONTROLE

A seguir será apresentada a tabela com valores necessários aos seus respectivos sinais, necessários para o funcionamento correto do datapath apresentado acima. A tabela mostra também os valores dos mesmos em cada uma das instruções suportadas.

CONTROL	R-type	LW	SW	BEQ	BNE	J	JAL	JR	I-type
RegDst	1	0	0	0	0	0	0	0	0
JR	0	0	0	0	0	0	0	1	0
Jump	0	0	0	0	0	1	1	1	0
ALUop	00	10	10	11	11	XX	XX	00	01
Branch	0	0	0	1	0	0	0	0	0
MemRead	0	1	0	0	0	0	0	0	0
MemtoReg	0	1	0	0	0	0	0	0	0
BNE	0	0	0	0	1	0	0	0	0
MemWrite	0	0	1	0	0	0	0	0	0
ALUsrc	0	1	1	0	0	0	0	0	1
RegWrite	1	1	0	0	0	0	1	0	1
JAL	0	0	0	0	0	0	1	0	0

ULA CONTROL	SLL	SRL	SRA	others
Shamt	1	1	1	0

Para compreender melhor o funcionamento da tabela, será dado um exemplo com as instruções do tipo R. Para estas instruções, o sinal RegDst deverá estar em 1, pois instruções do tipo R têm como registrador de destino, onde serão armazenados os resultados das operações, o registrador que se encontra no campo rd (instrução[15-11]) . Para que o datapath escreva no registrador correto, o mux deverá selecionar a segunda entrada. O sinal branch deverá estar em 0, pois operações de formato R não farão cálculos para determinar se haverá um pulo de instruções ou não. O sinal jump e JR deverão estar em 0, pois queremos o resultado do mux anterior, onde o novo valor de PC é calculado, seja apenas para avançar uma instrução ou pular para alguma outra, como em branchs. O sinal MemRead deverá estar em 0, pois não será lido nenhum valor da memória, porque em instruções do tipo R fazemos as operações de acordo com os valores contidos em registradores, apenas. O sinal MemtoReg deverá estar em 0, para o mux selecionar diretamente a saída da ULA, onde a operação foi feita e enviar o sinal para o banco de registrador, para então o resultado ser escrito no registrador de destino, selecionado em RegDst.

O sinal ALUOp compõe um único seletor para o controlador da ULA, que é o módulo que determinará qual operação a ULA deverá fazer. Então, para instruções do tipo R, ALUOp deverá estar em 00. Dessa forma, o controlador da ULA saberá que é uma operação aritmética que deverá ser feita, dependendo apenas do campo funct ([5-0]) da instrução que determina qual das operações aritméticas deverá ser executada. O sinal MemWrite deverá estar em 0, pois o resultado sempre será armazenado no registrador rd e não na memória quando estivermos fazendo operações do tipo R. O sinal ALUSrc deverá estar em 0, para que o mux selecione o valor do registrador rt como segundo operando de entrada na ULA. O sinal RegWrite deverá estar em 1, pois o resultado da ULA será armazenado no registrador rd (instrução [15-11]), sendo assim, haverá escrita em registrador.

Para as novas instruções solicitadas, SLL, SRL, SRA e outras, foi acrescido à ULA os devidos operadores juntamente ao campo SHAMT referente à instrução. Para isso, fez-se necessário criar uma nova entrada na ULA contendo os bits de 10 ao 6 da saída da memória de instruções.

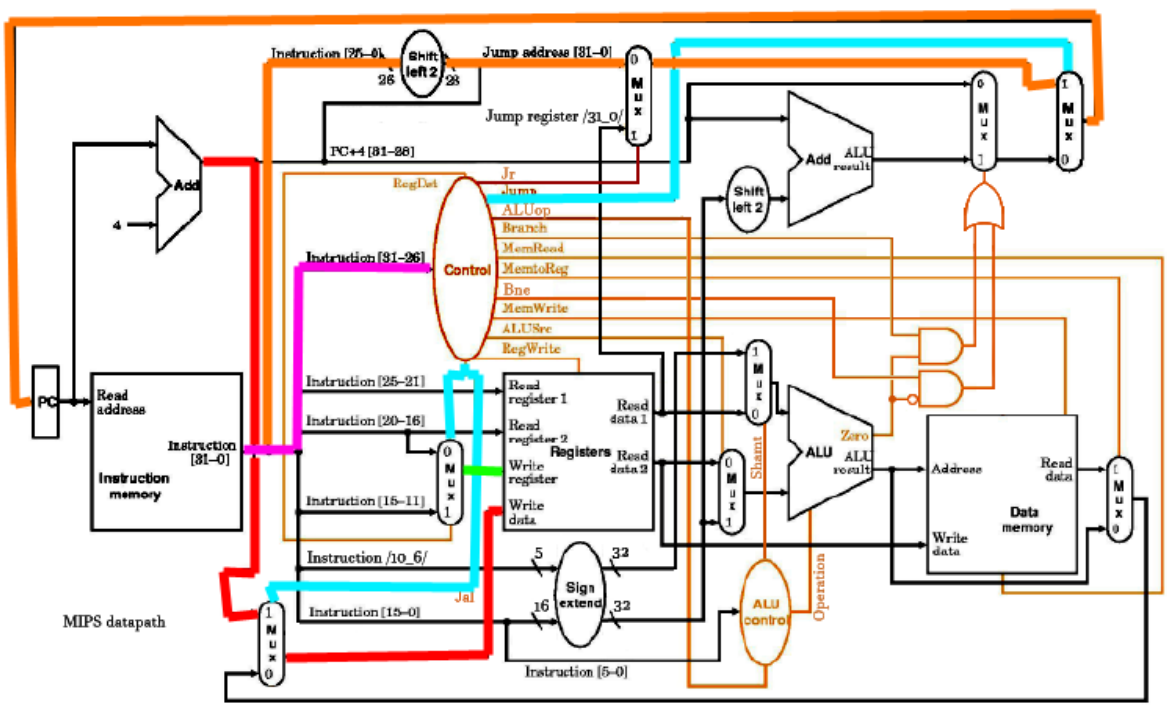
5. IMPLEMENTAÇÃO DAS INSTRUÇÕES JAL E JR

Instruction

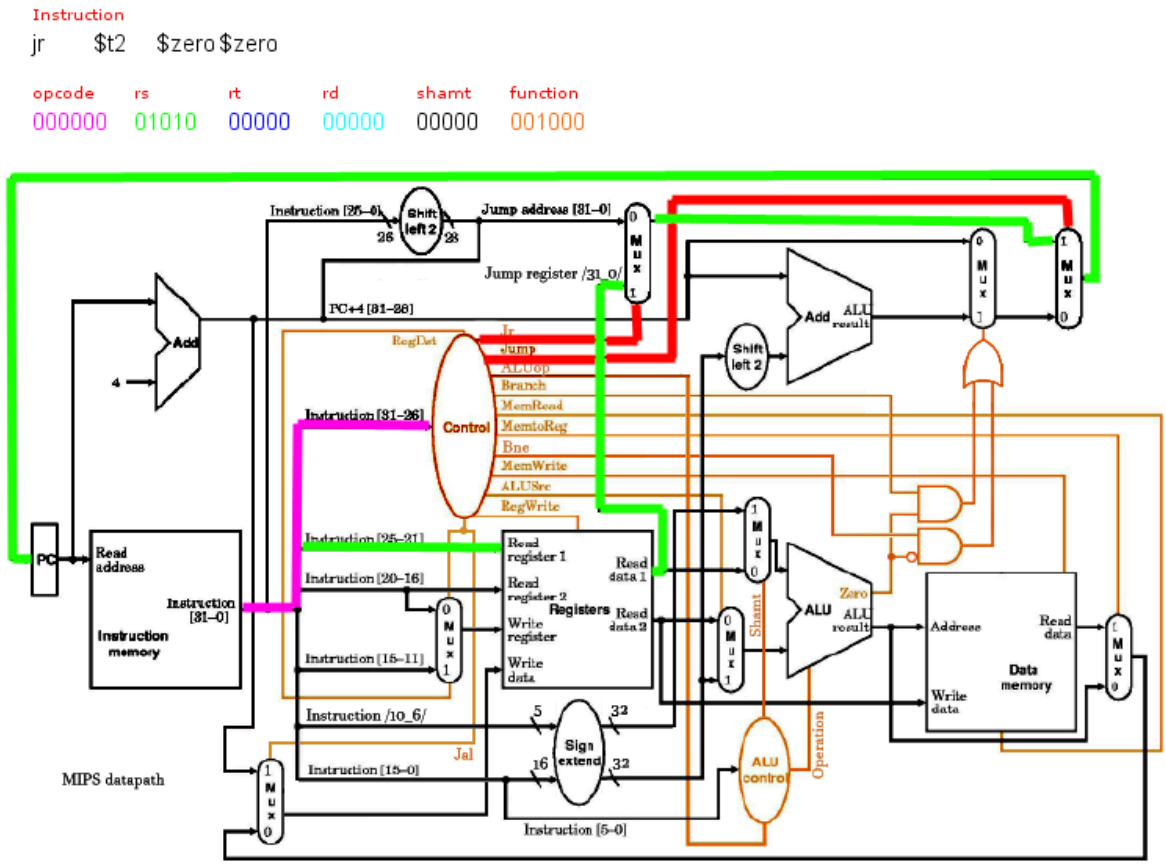
jal LABEL

opcode address

000011 00000000000000000000100010



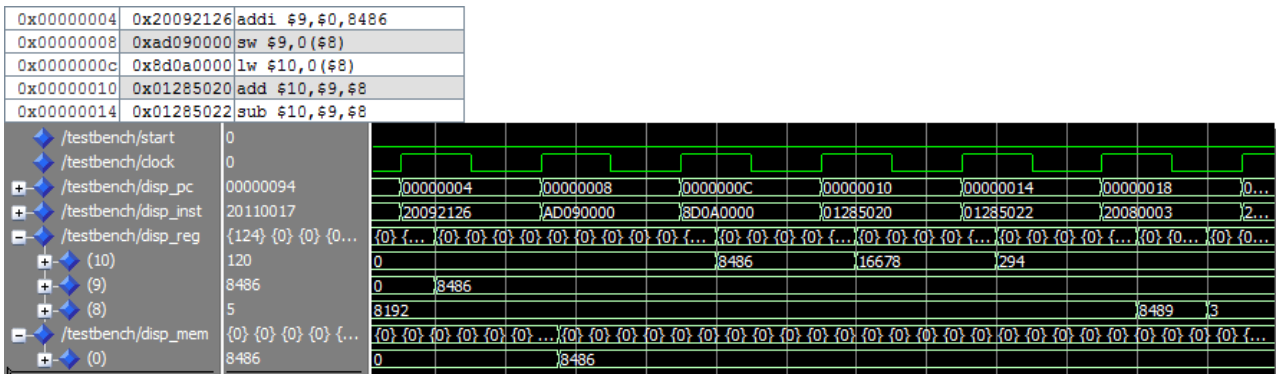
Para a instrução JAL, foi criado uma ligação a um mux2x1, ativado pela unidade de controle a partit da saída homônima. O sinal recebido faz com que a entrada WriteData (ou wdata no código VHDL) receba o sinal de PC+4. Esse sinal é guardado em um registrador que recebe o endereço 31, ou seja $ra. Esse sinal e gerado pelo mux de 5 bits, que ao receber o sinal de controle JAL, ignora as entradas principais, essas selecionadas através do sinal RegDst, que nesse caso é don't care. Já o endereço a ser assumido por PC sai direto do sinal da instrução vinda da memória, e passa por um shift left de 2 bits, e é incrementado com zeros em seus bits de maior índice. Após isso passa por 2 mux, um desativado para JR e um ativado pela saída Jump da unidade de controle até chegar no registrador PC.

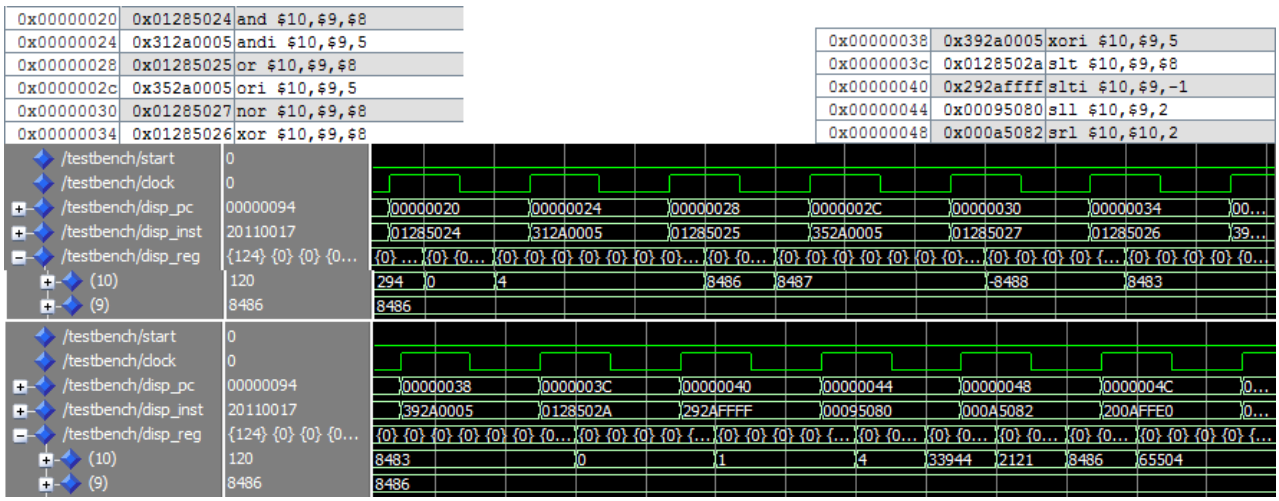


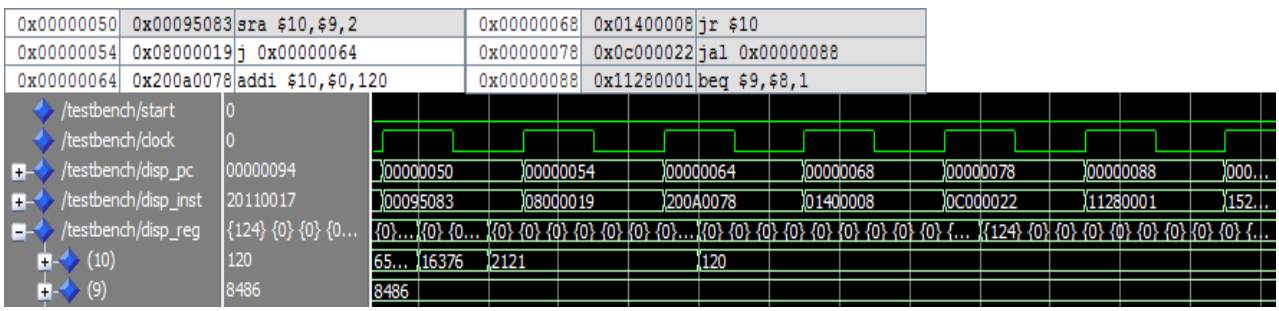
Por outro lado, quando o controle recebe uma instrução JR, esse sinal é ativado na saída juntamente com Jump e ativam os mux superiores para a passagem de um valor guardado em um registrador cujo endereço é obtido a partir da instrução decodificada. O sinal desse valor é na verdade o endereço a entrar no registrador PC.

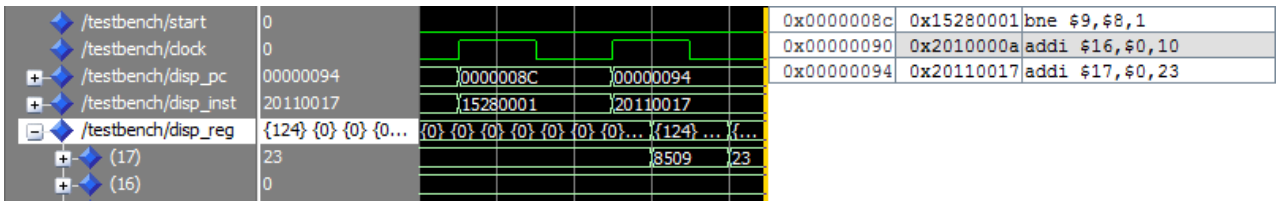
6. RESULTADOS

As figuras mostram o comportamento das instruções, submetidas a um clock em determinado intervalo, bem como informações de PC, memória e registradores. Vale ressaltar que, os dados válidos, que serão armazenados, são aqueles obtidos a cada fim de ciclo de clock, visto que durante todo o ciclo dados armazenados podem assumir diversos valores, porém só se assume como válido o valor ao fim do ciclo. A seguir serão apresentados os resultados obtidos para cada bloco de código, que foi utilizado pelo grupo como testbench.









7. CONCLUSÃO

A partir do desenho do datapath do MIPS – Uniciclo, que foi implementado utilizando a linguagem VHDL, e através do testbench, pode-se verificar que o processador funcionou da maneira esperada. As imagens apresentadas como resultado da simulação dos testes das instruções em assembly, demonstram que o comportamento está condizente com a teoria e os resultados se mostram satisfatórios.