LABORATÓRIO DE ARQUITETURA DE COMPUTADORES

Experimento 2

Registradores, Decodificação e Controle

(Banco de registradores, decodificação de instruções e sinais de controle)

GRUPO: X TURMA: K

Nome RA (ordem alfabética)

NOME DO ARQUIVO: Exp01\_G<nn> onde <nn> é o número do grupo

# Resumo (1/2 a 1 página)

O Experimento 2 baseou-se em prosseguir com a descrição, em VHDL, do processador MIPS simplificado. O objetivo do experimento foi adicionar, na parte inicial já descrita no Experimento 1 (módulo *Ifetch),* uma unidade de controle básica, uma unidade de decodificação e uma Unidade Lógica Aritmética (ULA) que, por enquanto, só realiza a execução da operação de soma.

Para a implementação dos módulos de controle e decodificação, foi necessário entender a estrutura das instruções de um processador MIPS, com seus devidos formatos. A instrução de soma é do tipo R-format, e, portanto, possui 6 bits para o código da operação, 5 bits para cada um dos três registradores (dois de origem e um de destino), 5 bits para possíveis deslocamentos, e 6 bits para a especificação da função. A unidade de controle (*control)* consistiu em habilitar um sinal que indica quando há escrita nos registradores, ou seja, está ativo alto todas as vezes que a instrução for do tipo R-format.

O módulo de decodificação (*Idecode)* separa os bits da instrução conforme foi descrito, e atribui a *ReadData1* e *ReadData2* os valores dos registradores de origem e a *ALU\_result* o valor a ser escrito no registrador de destino. Para a escrita, há um multiplexador que seleciona qual dos registradores de destino deve ser utilizado (*Rd* ou *Rt).* A instrução de soma, por ser do R-format, sempre utiliza o registrador *Rd*. Além disso, o *Idecode* descreve um *process que* inicializa os registradores com valores de 0 a 31 e permite a escrita na subida do *clock*. Finalmente, o módulo de execução (*execute)* é o qual realiza, de fato, a operação de soma.

O objetivo do experimento foi alcançado, dentro das limitações, visto que a implementação foi acompanhada da explicação teórica realizada em sala de aula.

# Código

Segue o código referente ao componente de decodificação *Idecode*:

LIBRARY IEEE;

USE IEEE.STD\_LOGIC\_1164.ALL;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

ENTITY Idecode IS

PORT(read\_data\_1 : OUT STD\_LOGIC\_VECTOR( 31 DOWNTO 0 );

read\_data\_2 : OUT STD\_LOGIC\_VECTOR( 31 DOWNTO 0 );

Instruction : IN STD\_LOGIC\_VECTOR( 31 DOWNTO 0 );

ALU\_result : IN STD\_LOGIC\_VECTOR( 31 DOWNTO 0 );

RegWrite : IN STD\_LOGIC;

RegDst : IN STD\_LOGIC;

Sign\_extend : OUT STD\_LOGIC\_VECTOR( 31 DOWNTO 0 );

clock,reset : IN STD\_LOGIC );

END Idecode;

ARCHITECTURE behavior OF Idecode IS

TYPE register\_file IS ARRAY ( 0 TO 31 ) OF STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL reg\_bank: register\_file;

SIGNAL write\_reg\_ID: STD\_LOGIC\_VECTOR(4 DOWNTO 0);

SIGNAL write\_data: STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL read\_Rs\_ID: STD\_LOGIC\_VECTOR(4 DOWNTO 0);

SIGNAL read\_Rt\_ID: STD\_LOGIC\_VECTOR(4 DOWNTO 0);

SIGNAL write\_Rd\_ID: STD\_LOGIC\_VECTOR(4 DOWNTO 0);

SIGNAL write\_Rt\_ID: STD\_LOGIC\_VECTOR(4 DOWNTO 0);

SIGNAL Immediate\_value: STD\_LOGIC\_VECTOR(15 DOWNTO 0);

BEGIN

-- Os sinais abaixo devem receber as identificacoes dos registradores

-- que estao definidos na instrucao, ou seja, o indice dos registradores

-- a serem utilizados na execucao da instrucao

read\_Rs\_ID <= Instruction(25 DOWNTO 21);

read\_Rt\_ID <= Instruction(20 DOWNTO 16);

write\_Rd\_ID <= Instruction(15 DOWNTO 11);

write\_Rt\_ID <= Instruction(20 DOWNTO 16);

Immediate\_value <= Instruction(15 DOWNTO 0);

-- Os sinais abaixo devem receber o conteudo dos registradores, reg(i)

-- USE "CONV\_INTEGER(read\_Rs\_ID)" para converser os bits de indice do registrador

-- para um inteiro a ser usado como indice do vetor de registradores.

-- Exemplo: dado um sinal X do tipo array de registradores,

-- X(CONV\_INTEGER("00011")) recuperaria o conteudo do registrador 3.

read\_data\_1 <= reg\_bank(CONV\_INTEGER(read\_Rs\_ID));

read\_data\_2 <= reg\_bank(CONV\_INTEGER(read\_Rt\_ID));

-- Crie um multiplexador que seleciona o registrador de escrita de acordo com o sinal RegDst

write\_reg\_ID <= write\_Rd\_ID WHEN RegDst = '1' ELSE write\_Rt\_ID;

-- Ligue no sinal abaixo os bits relativos ao valor a ser escrito no registrador destino.

write\_data <= ALU\_result;

-- Estenda o sinal de instrucoes do tipo I de 16-bits to 32-bits

-- Faca isto independente do tipo de instrucao, mas use apenas quando

-- for instrucao do tipo I.

Sign\_extend <= X"0000" & Immediate\_value

WHEN Immediate\_value(15) = '0'

ELSE X"FFFF" & Immediate\_value;

PROCESS

BEGIN

WAIT UNTIL clock'EVENT AND clock = '1';

IF reset = '1' THEN

-- Inicializa os registradores com seu numero

FOR i IN 0 TO 31 LOOP

reg\_bank(i) <= CONV\_STD\_LOGIC\_VECTOR( i, 32 );

END LOOP;

ELSIF RegWrite = '1' AND write\_reg\_ID /= "00000" THEN

reg\_bank(CONV\_INTEGER(write\_reg\_ID)) <= write\_data;

END IF;

END PROCESS;

END behavior;

Segue o assembly das instruções contidas em *program.mif* e o código de máquina referente às instruções:

ADD $t5 $zero $t7 #R13 = 0 + R15

ADD $t5 $zero $s0 #R13 = 0 + R16

ADD $t5 $zero $s3 #R13 = 0 + R19

ADD $t5 $zero $s6 #R13 = 0 + R22

ADD $t5 $t3 $t9 #R13 = R11 + R25

1. 0x000F6820
2. 0x00106820
3. 0x00136820
4. 0x00166820
5. 0x01796820

# Simulação e teste

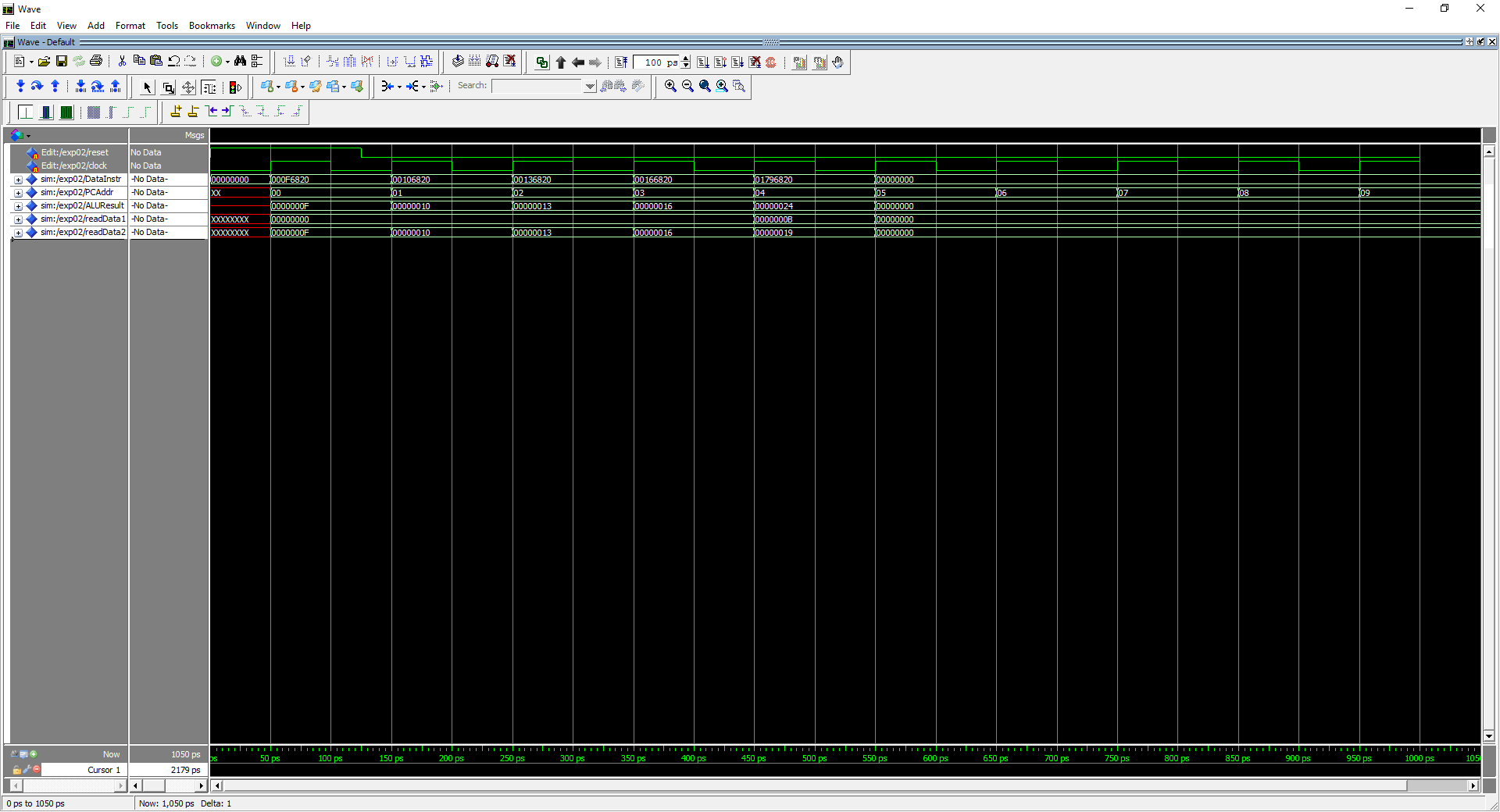


Fig. 1: Simulação.

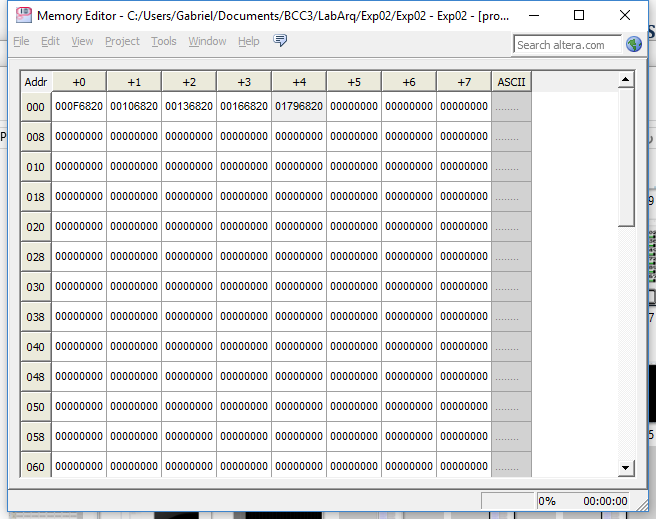


Fig 2. Conteúdo do arquivo *program*.*mif*.

Fig. 3: Estado dos registradores após a primeira instrução.

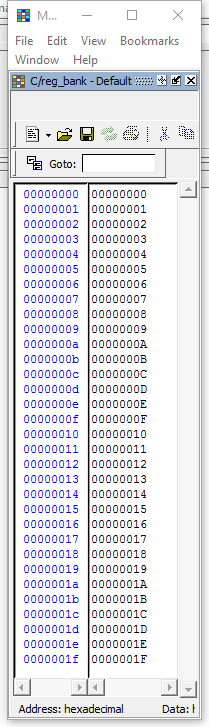
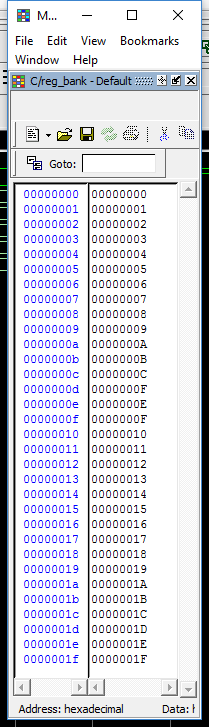
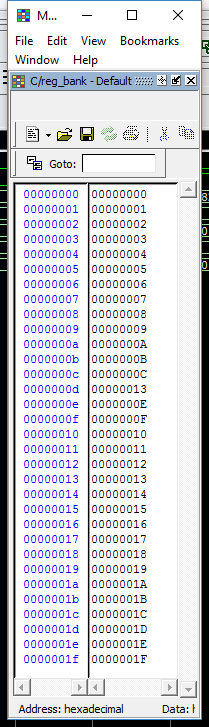


Fig. 4: Estado dos registradores após a segunda instrução.

Fig. 5: Estado dos registradores após a terceira instrução

Fig. 6: Estado dos registradores após a quarta instrução.

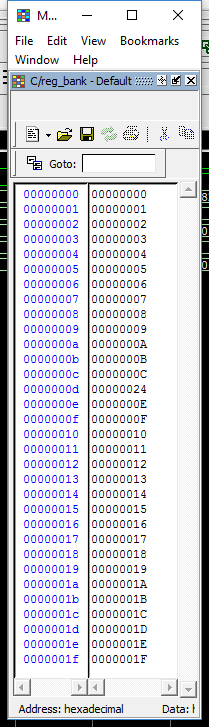
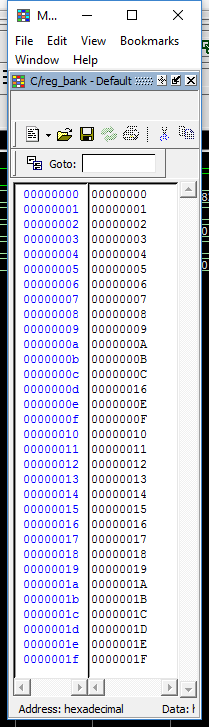


Fig. 7: Estado dos registradores após a quinta instrução.



## 

## 

## Discussão

Explicação pontual **RELACIONANDO** o código VHDL (item 2, referencie a linha de código ou operação) com as mudanças nos sinais de saída para a imagem de simulação apresentada (item 3), associe também ao conteúdo de memória. **Esta explicação deve deixar claro que a simulação representa o funcionamento correto do projeto descrito**.