LABORATÓRIO DE ARQUITETURA DE COMPUTADORES

Experimento 3

Memória de Dados,
Instrução Load e Store
E
Instrução de Salto Condicional

SUMÁRIO

1 Introdução teórica 1
1.1 Conjunto de Instruções 1
2 Instruções SW e LW 2
2.1 Componentes de projeto 2
2.2 Procedimento Preparatório 2
2.3 Pin Planner 3
2.4 Experimento 3
3 Instrução Beq 4
3.1 Procedimento Prático 5
3.2 Experimento 6
4 Resultados 7
5 Bibliografia 7

1 Introdução teórica

Este experimento é uma continuação do experimento anterior em direção à descrição do processador MIPS simplificado. Neste experimento vamos inserir a memória de dados e construir as instruções LW (Load Word) e SW (Store Word), depois iremos alterar o projeto para construir a instrução Beq (Branch if equal). A instrução LW lê uma palavra de um endereço de memória e guarda em um registrador, enquanto a instruções SW grava em um endereço de memória uma palavra armazenada em um registrador. Já a instrução BEQ realiza um desvio condicional na execução do programa, semelhante ao desvio condicional de execução realizado pela instrução *if* da linguagem C.

1.1 Conjunto de Instruções

A Tabela 1 apresenta a estrutura dos tipos de instruções e a Tabela 2 apresenta um conjunto reduzido de instruções para o processador MIPS.

Tabela 1 – Tipos de Instruções

Field Size	6-bits	5-bits	5-bits	5-bits	5-bits	6-bits
R - Format	Opcode	Rs	Rt	Rd	Shift	Function
I - Format	Opcode	Rs	Rt	Address/immediate value		
J - Format	Opcode	Branch target address				

Tabela 2 – Tipos de Instruções

Mnemonic	Format	Opcode Field	Function Field	Instruction
Add	R	0	32	Add
Addi	I	8	-	Add Immediate
Addu	R	0	33	Add Unsigned
Sub	R	0	34	Subtract
Subu	R	0	35	Subtract Unsigned
And	R	0	36	Bitwise And
Or	R	0	37	Bitwise OR
Sll	R	0	0	Shift Left Logical
Srl	R	0	2	Shift Right Logical
Slt	R	0	42	Set if Less Than
Lui	I	15	-	Load Upper Immediate
Lw	I	35	-	Load Word
Sw	I	43	-	Store Word
Beq	I	4	-	Branch on Equal
Bne	I	5	-	Branch on Not Equal
J	J	2	-	Jump
Jal	J	3	-	Jump and Link (used for Call)
Jr	R	0	8	Jump Register (used for Return)

2 Instruções SW e LW

As instruções LW e SW são do tipo I-format. As instruções do tipo I-format somam o endereço definido na instrução ao conteúdo do registrador de origem Rs. Para as instruções SW e LW, o registrador Rt é o registrador de origem e destino, respectivamente.

2.1 Componentes de projeto

Este experimento adiciona mais um componentes ao experimento 2: unidade de memória. Também faz alterações em outros componentes para que as instruções LW e SW possam ser criadas.

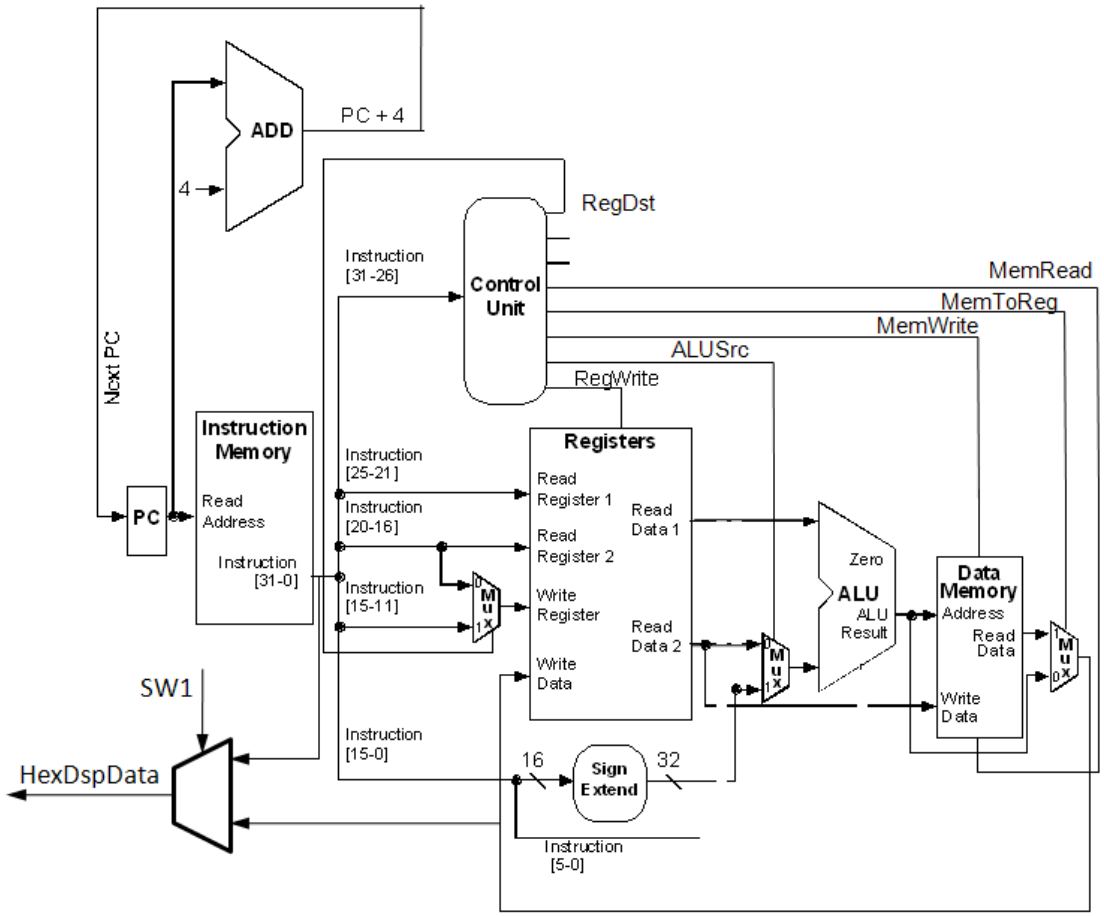

Figura 1 – Projeto MIPS parcial

2.2 Procedimento Preparatório

Agora vocês já sabem como declarar componentes e mapear os sinais de I/O dos componentes para interligá-los ao projeto. Então esta alteração na entidade TLE (Top Level Entity), quando é criado um novo componente, não estará mais explicita nos documentos dos experimentos, mas sempre serão necessárias.

- Crie um novo projeto de nome Exp03 e copie os arquivos disponíveis no moodle para a pasta do projeto.
- Selecione o menu “Assigments - Settings”, escolha a aba “Files” e clique em “Add All”, feche a janela. Isto fará com que o Quartus reconheça os arquivos copiados como parte do projeto e evitará alguns *warnings*. Com isto a lista dos arquivos poderá ser vista do lado esquerdo da janela escolhendo a guia “Files”.

Complemente os arquivos VHDL: Control, Idecode e Execute seguindo as orientações abaixo. Não se esqueça de editar o Exp03 conforme mostra a Figura 1.

- IDECODE: crie o multiplexador que seleciona para gravar no registrador o dado da memória ou da ULA. Edite a entrada WriteData para receber a saída do MUX.
- CONTROL: crie os sinais MenToReg, MenRead, MenWrite e ALUSrc. Será necessário editar outros já existentes. Pense quais sinais precisam ser ativados para as instruções novas.
- EXECUTE: crie o multiplexador que seleciona entre read_data_2 e SignExtend como entrada do somador.
- DMEMORY: esta entidade é fornecida. Olhe o código para entender o tipo de memória de dados definida. Será necessário defini-la em Exp03 e fazer o *port map*. O arquivo dmemory.mif define o conteúdo inicial da memória.

```
--  Configuracao da memoria
DEPTH = 256;           % valor decimal com numero de enderecos %
WIDTH = 32;            % valor decimal com numero de bits da palavra %
-- Configuracao da apresentacao
ADDRESS_RADIX = HEX;       % Opcional: endereco em formato hexadecimal %
DATA_RADIX = HEX;          % Opcional: dado em formato hexadecimal %
-- Define o conteudo inicial da memoria
Content
Begin
-- atribui valores a enderecos especificos
 00 : 8C020000;       % atribui dado hexa 8C020000 para endereco 00 %
 01 : 8C030001;       % atribui dado hexa 8C030001 para endereco 01 %
 02 : 00430820;       % atribui dado hexa 00430820 para endereco 02 %
 03: AC010003;
 04: 1022FFFF;
 05: 1021FFFA;
-- valores defaut, preenche memoria com zeros
 [06..FF] : 00000000;
End;
```

Figura 2 – Estado Inicial de Memória

2.3 Pin Planner

Após finalizado o projeto importe a atribuição de pinos através do menu *Assignments – Import Assignments* e escolha o arquivo Exp03.csv. Abra o PinPlanner e verifique se está tudo certo. Então configure os pinos não utilizados como “As input tri-state”.

NÃO SE ESQUEÇA DE COMPILAR O PROJETO APÓS ESTA CONFIGURAÇÃO.

2.4 Experimento

Implemente a descrição no simulador Quartus II e compile. Simule e teste o modelo descrito para verificar se o comportamento é como o esperado.

SIMULAÇÃO (usar RTL Simulation)

- Crie um sinal de clock com período de 100pS e duty cycle de 50%, início baixo.

- Crie um sinal de reset que inicia ativo (=1) e permanece até 75pS, indo para zero até o final da simulação. Este intervalo de tempo é necessário para a simulação de leitura de memória.
- Adicione os sinais de saída que deseja monitorar.
- Use a opção de visualizar "Memory list" para acompanhar as alterações de memória e registradores.

TESTE

Somente se a simulação for bem sucedida, faça upload para a placa após o *check list*. Para teste na placa faça:

- Ative o reset, dê um pulso de clock com o reset ativo, após desative o reset.
- O conteúdo da primeira posição de memória deve aparecer no display de LCD.
- Mude a chave SW1 para mostrar o resultado da soma, retorne para ver a instrução. Confira se o resultado é o esperado para aquela instrução, considerando o valor dos registradores naquele momento.
- Pulse o clock para ler as demais posições.

3 Instrução Beq

BEQ é uma instrução do tipo I-Format. Esta instrução compara o conteúdo dos registradores Rs e Rt e, se forem iguais, a execução do programa é desviada para o **endereço relativo** definido pelos bits menos significativos da instrução (15 a 0). Endereço relativo significa que o desvio ocorre em relação à posição atual de execução (próxima instrução). Os 16 bits menos significativos indicam de quantas instruções será o desvio para mais ou menos em relação ao (PC+1)[1], que aponta para a próxima instrução. Como o valor de desvio pode ser positivo ou negativo, é representado em complemento de 2 com sinal. Por exemplo, observe as instruções `beq` no código de máquina abaixo:

```
00: 8C020000;     -- lw $2,0 ;memory(00)=55
01: 8C030004;     -- lw $3,4 ;memory(01)=AA
02: 00430820;     -- add $1,$2,$3
03: AC010003C;    -- sw $1,12 ;memory(03)=FF
04: 1022FFFF;     -- beq $1,$2,-4
05: 1021FFFA;     -- beq $1,$1,-24
```

Na instrução `beq $1,$2,-4 (1022FFFF)`[2], se o conteúdo dos registradores $1 e $2 forem iguais, o salto ocorrerá para a própria instrução, pois FFFF corresponde ao valor decimal -1 (complemento de 2 com sinal) e (PC+1) aponta para a próxima instrução (endereço 05). Fazendo (PC+1) + (-1), o contador de programa passará a apontar para o endereço 04. Neste caso o programa ficará parado. Observe que tanto o valor a ser somado quanto o (PC+1) operam sobre uma palavra, de 4 bytes.

De forma equivalente, podemos concluir que a instrução `beq $1,$1,-24 (1022FFFA)` retorna o contador de programa para o início do programa, no endereço 00. Observe que FFFA corresponde ao valor decimal -6. Neste momento PC+1 aponta para o endereço 06. Fazendo (PC+1) + (-6) teremos o endereço 00. A instrução em *assembly* indica o desvio relativo como sendo -24 porque esta contagem está em bytes, mas em código de máquina contém o valor em palavras de 4 bytes, portanto -06.

[1] (PC+1) refere-se à próxima instrução, pois +1 indica o número de palavras, sendo que cada instrução tem 4 bytes.

[2] Observe que em assembly o valor a ser somado é dado em bytes e em linguagem de máquina é dado em palavras/instruções (4 bytes).

3.1 Procedimento Prático

A Figura 1 ilustra as alterações necessárias no circuito para criar a instrução BEQ. Se o sinal de controle indicar que é uma instrução do tipo *Branch* e **(AND)** o resultado da ULA indicar que o conteúdo dos registradores Rs e Rt são iguais (*zero*), o NextPC deve ser o resultado da soma do PC+1 (words) com o signExtend (words), caso contrário o NextPC deve ser o PC+1. Então temos as seguintes alterações serem feitas sobre o experimento anterior para obter o esquema da Figura 1:

- Criação do sinal de controle Branch (entidade CONTROL), que é ativo alto sempre que for uma instrução do tipo beq;
- Criação do sinal Zero da ULA (entidade EXECUTE), que é ativo alto sempre que Rs=Rt;
- Criar o circuito ADD (entidade EXECUTE), que deve somar o endereço do PC+1 (words) com o SigExtend (words), gerando sinal ADDResult;
- Criar o multiplexador (entidade IFETCH) que decide se o fluxo de execução do programa segue o curso normal (NextPC <= PCInc = PC+1) ou desvia (NextPC <= ADDResult), se os sinais Branch E Zero forem ativos.
- Ajuste as entradas e saídas (IN, OUT) das entidades alteradas acima para fazer as novas ligações através do port map.
- Façam Ctr-C e Ctr-V das novas definições de PORT das entidades CONTROL, EXECUTE e IFETCH para a declaração dos componentes na entidade TLE (Exp04).
- Faça as ligações "Port map" necessárias na TLE.
- Importe os pinos de I/O do arquivo Exp04.csv e defina os pinos não utilizados como "Input tri-state". Compile.

Observe a o código VHDL deve ser alterado para obter o fluxo de dados marcado pela região em vermelho. Agora o NextPC não mais recebe o sinal PC+1, mas sim a saída de um multiplexador (MUX) que escolhe entre PC+1 e o endereço de salto da instrução BEQ.

Observação: nos livros o SignExtend é multiplicado por 4 (shift left 2), porque a memória e PC são referenciados em bytes. Como estamos usando endereçamento por palavras, somamos diretamente o PC+1 (PCinc em palavras) com o SignExtend que já está em palavras.

```
-- MIPS Instruction Memory Initialization File
Depth = 256;
Width = 32;
Address_radix = HEX;
Data_radix = HEX;
Content
Begin
    00: 8C020000;     -- lw $2,0 ;memory(00)=55
    01: 8C030004;     -- lw $3,4 ;memory(01)=AA
    02: 00430820;     -- add $1,$2,$3
    03: AC01000C;     -- sw $1,12 ;memory(03)=FF
    04: 8C04000C;     -- lw $4,3 ;memory(03)=FFFFFFFF
    05: 1022FFFF;     -- beq $1,$2,-4
    06: 1021FFF9;     -- beq $1,$1,-28
   [07..FF]: 00000000;
End;
```

Figura 3 – Estado Inicial de Memória

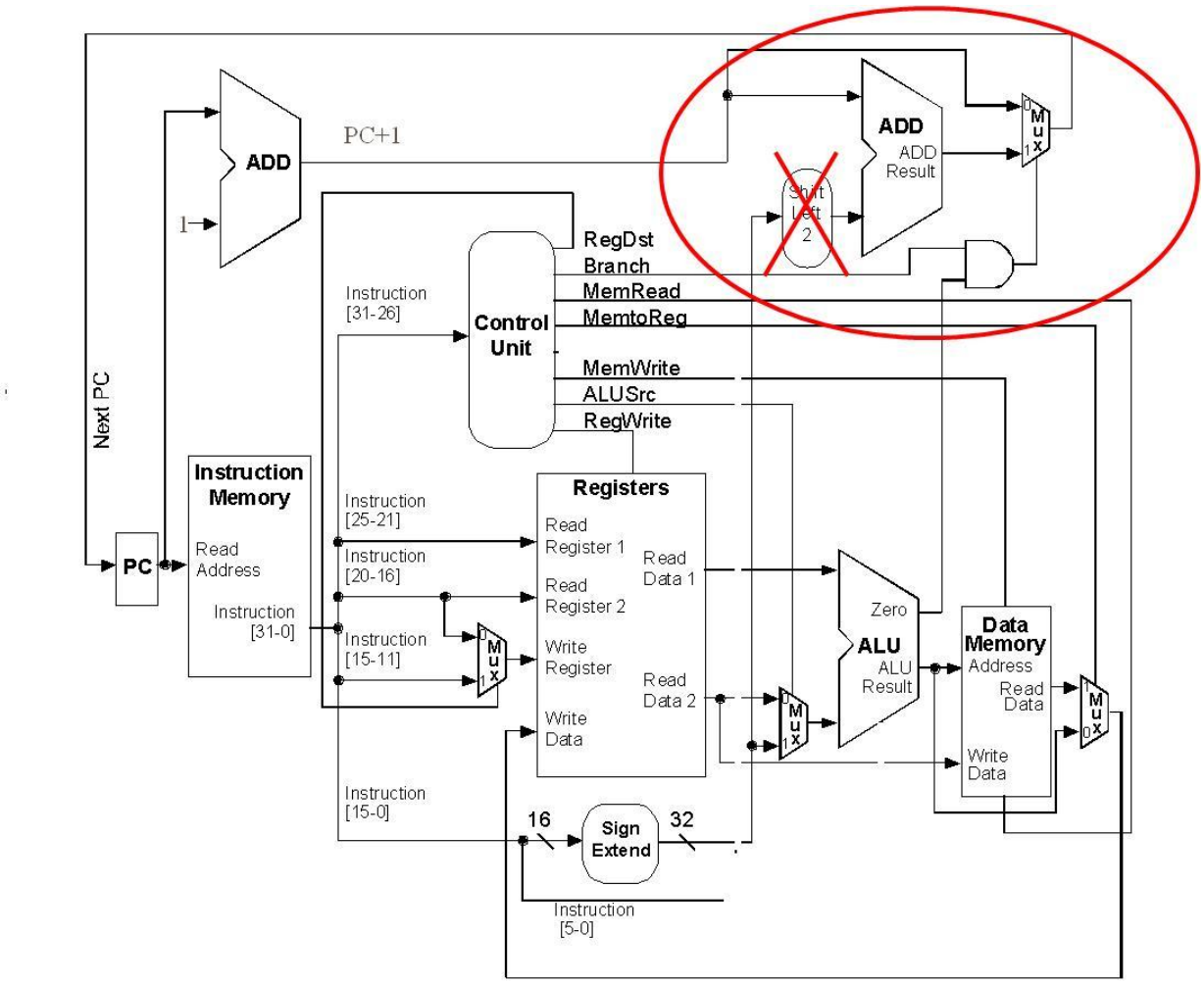

Figura 4 – Projeto MIPS parcial

3.2 Experimento

Crie um novo projeto de nome Exp04 e importe o código finalizado do experimento 3. Complemente as entidades VHDL, descrevendo o projeto conforme apresentado no item 3.1 e na Figura 1.

SIMULAÇÃO (Use RTL Simulation)

- Crie um sinal de clockPB e reset, sendo o reset com duração mínima para ter uma descida do clock.
- Adicione os sinais de saída que precisa monitora para validar a instrução.

TESTE

Somente se a simulação for bem sucedida, faça upload para a placa após o *check list*. Para teste na placa faça:

- Ative o reset, dê um pulso de clock com o reset ativo, após desative o reset.
- O conteúdo da primeira posição de memória deve aparecer no display de LCD.
- Mude a chave SW1 (InstrALU) para mostrar o resultado da soma ou memória, retorne para ver a instrução. Confira se o resultado é o esperado para aquela instrução, considerando o valor dos registradores naquele momento.
- Pulse o clock para ler as demais posições.

4 Resultados

Relatório

Faça um relatório (arquivo pdf) e entregue até a data definida através do AVA, juntamente com os arquivos VHDL das entidades alteradas (IFETCH, EXECUTE E CONTROL). O relatório deve conter:

- Identificação do experimento e grupo conforme modelo.
- Resumo do experimento (1/2 a 1 página)
- Imagem do simulador com as ondas de simulação e teste.
- Explicação pontual sobre o instante de tempo da imagem de simulação apresentada, **RELACIONANDO** a instrução da memória de programa com as mudanças nos sinais de saída. Esta explicação deve deixar claro que a simulação representa o funcionamento correto do código de máquina (instruções).

Apresentação

- Apresentar o funcionamento na placa para a professora na próxima aula.

5 Bibliografia

D'Amore, R. VHDL: Descrição e Síntese de Circuitos Digitais. LTC. 2005.

Hamblen, J.O; Hall, T.S. & Furman, M.D – Rapid Prototyping of Digital Systems: SOPC Edition. Springer, 2008.