LABORATÓRIO DE ARQUITETURA DE COMPUTADORES

Experimento 2

Registradores, Decodificação e Controle

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

SUMÁRIO

1 Introdução teórica

Este experimento é uma continuação do experimento anterior em direção à descrição do processador MIPS simplificado. Neste experimento vamos inserir os registradores internos do processador e iniciar a descrição da unidade de controle para decodificar as instruções e executar uma das instruções de soma.

Para implementar a decodificação das instruções é necessário primeiro conhecer as estruturas possíveis para as instruções no MIPS. O MIPS possui instruções de tamanho fixo de 32 bits e 32 registradores de propósito geral de 32 bits cada um. O registrador 0 sempre possui valor zero e o tamanho da palavra de memória é de 32 bits.

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 – Conjunto Reduzido 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)

As instruções do tipo R realizam as operações sobre os registradores Rs e Rt (origem de dados) e armazenam o resultado no registrador Rd (destino dos dados). Enquanto as instruções do tipo I usam o registrador Rt como destino. Já as instruções do tipo J não utilizam explicitamente um registrador.

Observe na Tabela 1 que todas as instruções iniciam com 6 bits (31 a 26) de campo de *opcode*, utilizado para identificar a instrução. A Tabela 2 mostra que no caso da instrução do tipo R, o *opcode* é sempre zero e o campo *function*, também de 6 bits (5 a 0), indica a operação a ser realizada.

1.2 Componentes de projeto

Este experimento adiciona 3 componentes ao experimento 1: unidade de controle, decodificação e execução (apenas soma). A Figura 1 e Figura 2 mostram os componentes (entidades) de controle (*Control*) e decodificação (*Idecode*), enquanto a Figura 3 mostra o componente da Unidade Lógica Aritmética (*Execute*), que deve ser executar apenas a soma (*add*). A unidade de controle deve gerar os sinais de controle para a unidade de decodificação.

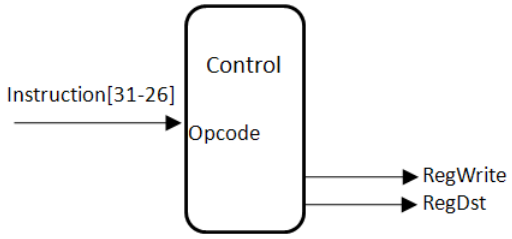

Figura 1 – Fase 1: Unidade de Controle

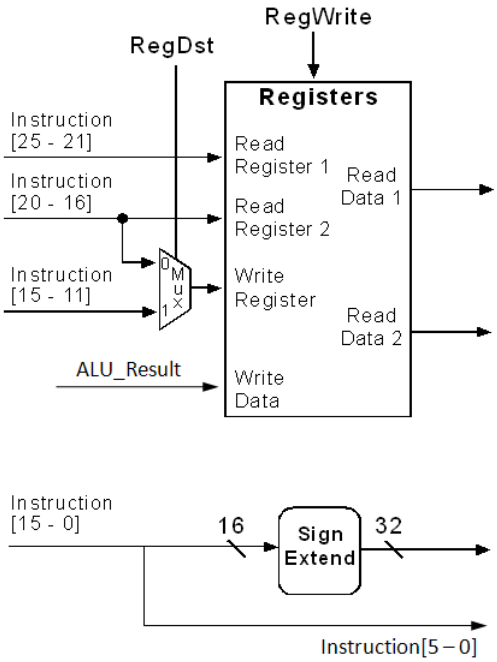

Figura 2 – Fase 1: Componente Decodificação (Idecode)

A Figura 2 mostra a unidade de decodificação (Idecode) que separa a instrução em partes para **identificar** os registradores de origem (*ReadRegister1* e *ReadRegister2*) e destino (*WriteRegister*). Após decodificar a instrução as saídas *ReadData1* e *ReadData2* devem conter o **conteúdo** dos registradores de origem identificados. O sinal de entrada *ALU_Result*, que será definido pela unidade de execução, fornecerá o valor (*WriteData*) a ser guardado no registrador de destino.

O sinal ***RegWrite*** deve ser **ativo alto SEMPRE QUE a instrução exigir escrever um valor no registrador**. Uma instrução do tipo R-format sempre escreve o resultado da operação no registrador. O registrador de destino pode ser Rt ou Rd dependendo do tipo de instrução, então deve ser colocado um multiplexador para selecionar o registrador correto através do sinal de controle *RegDst*. O registrador destino será Rd somente quando a instrução for do tipo *R-format*.

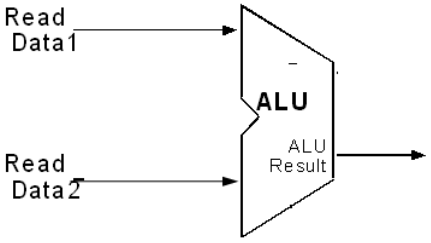

Figura 3 – Unidade lógica e aritmética (Execute): somador

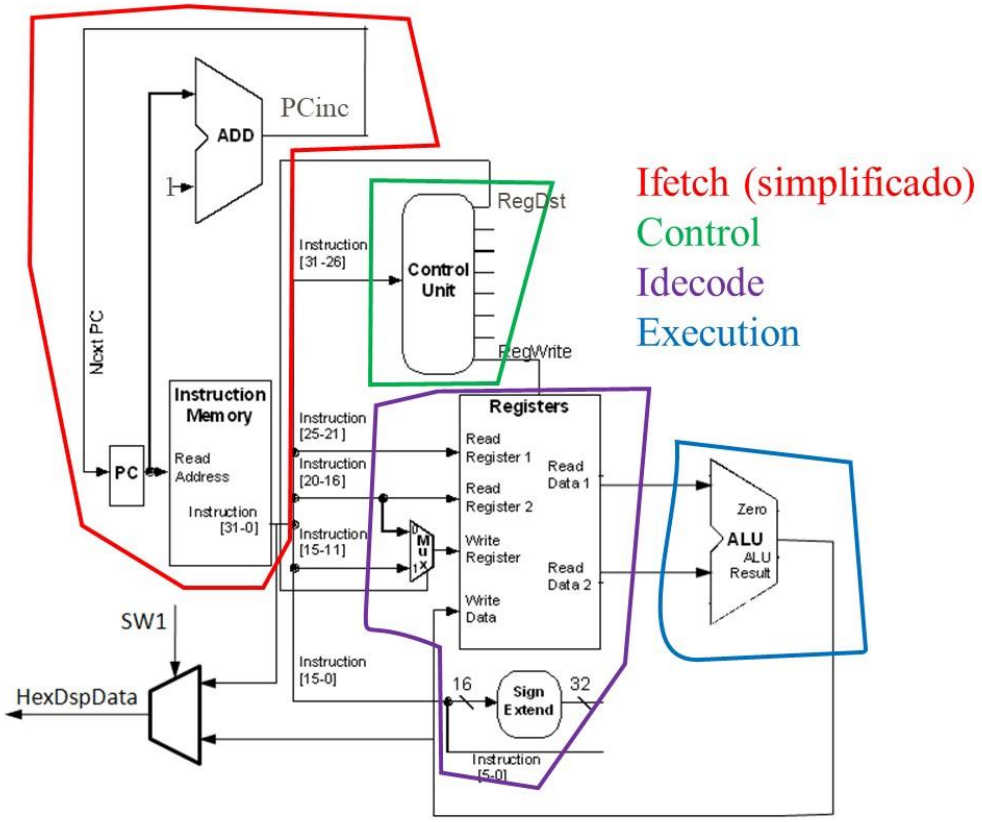

Figura 4 – Projeto MIPS parcial

A Figura 4 mostra a interligação dos componentes (entidades) para formar o MIPS nesta fase. Um multiplexador foi adicionado na saída a ser ligada no display de LCD para poder selecionar a visualização da instrução ou do resultado da Unidade Lógica Aritmética (ALU_Result), ou seja, até o momento, da soma.

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 com TLE de nome Exp02;
- Na janela "Add Files", faça uma pausa e copie os arquivos disponíveis no AVA para a pasta do projeto. Copie também os arquivos vhd do Experimento 01: **Ifetch, LCD_Display,** para a pasta do projeto;
- Verifique se o port map do iftetch no Exp02.vhd está correto com seu ifetch do experimento 1;
- Depois clique no botão "Add All" para que o Quartus reconheça os arquivos como parte do projeto;
- Importe o mapeamento dos pinos de I/O do arquivo Exp02.csv disponível;
- Configure os pinos não utilizados como "As Input tri-state";
- Complemente os arquivos VHDL: *Exp02, Control*, *Idecode* e *Execute* seguindo as Figuras de 1 a 4 e as instruções abaixo. Não se esqueça de editar o Exp02 conforme mostra a Figura 4.

2.1 Controle

Editar esta entidade tal que o sinal *RegWrite* deve ser ativo alto quando a instrução exigir uma escrita nos registradores, ou seja, sempre que for uma instrução do tipo R. O sinal *RegDst* deve ser ativo alto sempre que o registrador destino for Rd, ou seja, sempre que for uma instrução do tipo R. Usar o sinal interno R_format para sinalizar que é uma instrução do tipo R_format.

2.2 ULA

A Unidade Lógica Aritmética deve realizar apenas a operação de soma sobre o conteúdo dos registradores de origem (Rs e Rt) indicados pela instrução ADD e armazenar o resultado no registrador destino (Rd). A unidade Idecode é responsável por colocar as informações corretas em Read_data_1 e Read_data_2. E o resultado da soma, em ALU_Result, deve ser enviado à unidade Idecode para armazenamento no registrador Rd. Observe o *import* das bibliotecas IEEE, necessárias para construir um somador binário utilizando o sinal +.

2.3 Decodificador

O arquivo Idecode.vhd fornece os sinais do *PORT* e as indicações dos sinais que precisam ser gerados, mas não declara os sinais internos da entidade.

- Primeiro separe a instrução em partes de acordo com os tipos de instrução possíveis (R_format, I_format), que utilizam registradores. Obtenha o número que identifica os registradores usados como origem (Rs e Rt) e destino (Rd ou Rt), assim como o "valor imediato" que pode ser utilizado por instruções do tipo I-format. Use os sinais já definidos com nomes xx_ID para indicar a identificação do registrador.
- O conteúdo dos registradores de origem deve ser recuperado para as saídas read_data_1 e read_data_2. Use "CONV_INTEGER(Rs_ID)" para converter o ID do registrador no formato de bits para um valor inteiro que possa ser utilizado como índice do vetor de registradores.
- Um multiplexador deve escolher através do sinal de seleção RegDst o número do registrador destino, dependendo do tipo de instrução. Se for instrução do tipo R-format (RegDst='1'), o registrador destino deve ser Rd, caso contrário o registrador destino deve ser Rt.
- Observe que o bloco "Sign Extend" tem uma entrada de 16 bits e saída de 32 bits. Isto significa que o **sinal deve ser estendido para ter 32 bits**. Mas este sinal pode

ser **positivo ou negativo**, logo deve ser usada a **regra de extensão binária com sinal** vista em Circuitos Digitais.

- Por último é necessário descrever o circuito que escreve nos registradores. O registrador somente é escrito na subida do clock, quando a instrução finaliza para a decodificação de outra. Portanto, a escrita no registrador deve ser descrita como um *process*. Ao *resetar* o sistema os registradores devem ser iniciados com valor igual a sua numeração, por exemplo, R0=0, R1=1, etc. Após o reset os registradores somente são escritos através das instruções que assim exigem. O sinal *RegWrite* deve ser usado para sinalizar esta condição. O registrador zero R(0) nunca deve ser sobrescrito, portanto o código VHDL que escreve nos registradores deve verificar esta condição.

2.4 TLE (Exp02)

Interligue os componentes novos (Control, Idecode e Execute) à entidade principal, crie e use sinais internos (SIGNAL) para isto, conforme Figura 4. **Adicione um multiplexador à entidade TLE (Exp02) tal que possa selecionar o que apresentar no display de LCD: instrução ou resultado da soma (ALUResult)**. Adicione um sinal de I/O (PORT) para o InstrAlu (SW1) como seletor deste multiplexador.

2.5 Pin Planner e Unused Pins

Caso não tenha feito na criação do projeto: após finalizado o projeto importe a atribuição de pinos através do menu *Assignments – Import Assignments* e escolha o arquivo Exp02.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.

2.6 Memória Inicial

Um arquivo contendo o estado inicial da memória deve ser definido. No caso do experimento foi definido um arquivo de nome ***program.mif***. A Figura 5 mostra o arquivo para definição do estado inicial de memória deste experimento. Você deve entender as instruções ADD nos endereços 00 a 03 e definir a instrução no endereço 04 tal que some o conteúdo dos registradores R11 e R25 e guarde o resultado no registrador R13.

```
--  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  :  000F6820;
    01  :  00106820;
    02  :  00136820;
    03  :  00166820;
    04 : <ALTERE ESTA INSTRUÇÃO PARA QUE  FAÇA R13=R11+R25>
-- valores defaut, preenche memoria com zeros
 [05..FF] : 00000000;
End;
```

Figura 5 – Estado Inicial de Memória

3 Experimento

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

SIMULAÇÃO

- Crie um sinal de clock com período de 100pS e duty cycle de 50%, início alto.
- Crie um sinal de reset que inicia ativo (=1) e permanece até 75 pS, indo para zero até o final da simulação. Este intervalo de tempo é necessário para a simulação de leitura de memória para o endereço zero.
- Adicione os sinais de saída que deseja monitorar.

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. Isto deve ser feito para que o conteúdo do endereço zero seja colocado na saída da memória.
- O conteúdo da primeira posição de memória deve aparecer no display de LCD.
- Mude a chave SW1 (InstrAlu) 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.

4 Resultados

Relatório

Faça um relatório e entregue até a data definida, através do moodle em arquivo pdf.

- Identificação do experimento e grupo conforme modelo.
- Código VHDL da entidade IDecode.
- Assembly das instruções contidas em program.mif e o código de máquina da instrução criada.
- Imagem do simulador com as ondas de simulação e teste.
- Explicação pontual **RELACIONANDO** o código VHDL e instrução da memória com as mudanças nos sinais de saída para a imagem de simulação apresentada. Esta explicação deve deixar claro que a simulação representa o funcionamento correto do código VHDL descrito.

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.