Trabalho Prático 2 – MIPS

Organização de Computadores I (DCC006)
Antônio Otávio Fernandes e Lucas Ferreira de Melo Diniz

O Mips

Como vocês aprenderam na disciplina, um processador pode ser dividido em etapas de forma a otimizar a utilização de recursos e aumentar o *throughput*. Por isso, é importante conhecer cada uma dessas etapas. Como já devem saber, os estágios de *Pipeline*: são:

- Fetch
- Decode
- Execute
- Memory
- Writeback

Além desses estágios é importante conhecer também alguns módulos importantes para o funcionamento do processador: MemController, RAM e o Banco de Registradores.

O objetivo desse trabalho é fazer com que todos conheçam o funcionamento de todas as etapas de *Pipeline*, possam juntá-las e finalmente testar o comportamento final do processador. A implementação e a especificação de cada fase será disponibilizada e fica a cargo de cada aluno validar o comportamento de cada uma das fases e do processador final.

Os módulos necessários para a contrução e teste do MIPS foram disponibilizadas e a seguir a especificação de cada uma delas, listando a funcionalidade de cada sinal presente no módulo.

Fetch

A unidade de busca de instruções é responsável por receber uma instrução do controlador de memória e repassa-la ao estágio de decodificação. Além disso, o contador de programa (PC) é controlado nesse módulo.

```
module Fetch (
    input                   clock,
    input                   reset,
    //Execute
    input                   ex_if_stall,
    //Decode
    output reg    [31:0]    if_id_nextpc,
    output reg    [31:0]    if_id_instruc,
    input                   id_if_selpcsource,
    input         [31:0]    id_if_rega,
    input         [31:0]    id_if_pcimd2ext,
    input         [31:0]    id_if_pcindex,
    input         [1:0]     id_if_selpctype,
    //Memory Controller
    output reg              if_mc_en,
    output        [31:0]    if_mc_addr,
    input         [31:0]    mc_if_data
);
```

Sinal	Função
clock	Sinal de clock para controle do acesso. As operações no estágio de busca de instruções são realizadas na subida do clock
reset	Sinal de reset assíncrono que coloca todos os registradores deste módulo em 0 na borda de descida do mesmo.
ex_if_stall	Sinal recebido do estágio de execução que indica que a instrução realizará um acesso à memória e, portanto, o estágio de busca de instruções deve inserir um *stall* no pipeline
if_id_nextpc	Recebe o mesmo valor que PC, a não ser que tenha havido um *stall*, situação em que o PC anterior é atribuído à esse sinal
if_id_instruc	Instrução recebida do controlador de memória caso um *stall* não tenha sido inserido, situação na qual a instrução NOP (32'd0) deve ser atribuída ao sinal
id_if_selpcsource	Indica que PC receberá o valor definido por id_if_selpctype, quando em 1, ou PC + 4, quando em 0
id_if_rega	Um dos possíveis valores que pode ser atribuído ao PC, dependendo de id_if_selpctype
id_if_pcimd2ext	Um dos possíveis valores que pode ser atribuído ao PC, dependendo de id_if_selpctype
id_if_pcindex	Um dos possíveis valores que pode ser atribuído ao PC dependendo de id_if_selpctype
id_if_selpctype	Seleciona o valor a ser atribuído ao PC, caso id_if_selpcsource seja 1, de acordo com a seguinte tabela: • 00: id_if_pcimd2ext • 01: id_if_rega • 10: id_if_pcindex • 11: 32'd64
if_mc_en	Indica que uma instrução deve ser lida da RAM
if_mc_addr	Indica onde a próxima instrução deve ser lida, ou seja, recebe PC
mc_if_data	Instrução lida da RAM

Decode

A unidade de decodificação é responsável por receber uma instrução do estágio de busca de instruções, decodifica-la determinando o valor dos sinais de controle e acessar os registradores a serem utilizados como operandos. Instruções de *branch* e *jump* também são resolvidas neste estágio.

```
module Decode (
    input                   clock,
    input                   reset,
    //Fetch
    input         [31:0]    if_id_instruc,
    input         [31:0]    if_id_nextpc,
    output                  id_if_selpcsource,
    output        [31:0]    id_if_rega,
    output        [31:0]    id_if_pcimd2ext,
    output        [31:0]    id_if_pcindex,
    output        [1:0]     id_if_selpctype,
    //Execute
    output reg              id_ex_selalushift,
    output reg              id_ex_selimregb,
    output reg    [2:0]     id_ex_aluop,
    output reg              id_ex_unsig,
    output reg    [1:0]     id_ex_shiftop,
    output        [4:0]     id_ex_shiftamt,
    output        [31:0]    id_ex_rega,
    output reg              id_ex_readmem,
    output reg              id_ex_writemem,
    output        [31:0]    id_ex_regb,
    output reg    [31:0]    id_ex_imedext,
    output reg              id_ex_selwsource,
    output reg    [4:0]     id_ex_regdest,
    output reg              id_ex_writereg,
    output reg              id_ex_writeov,
    //Registers
    output        [4:0]     id_reg_addra,
    output        [4:0]     id_reg_addrb,
    input         [31:0]    reg_id_dataa,
    input         [31:0]    reg_id_datab,
    input         [31:0]    reg_id_ass_dataa,
    input         [31:0]    reg_id_ass_datab
);
```

Sinal	Função
clock	Sinal de clock para controle do acesso. As operações no estágio de busca de instruções são realizadas na subida do clock
reset	Sinal de reset assíncrono que coloca todos os registradores deste módulo em 0 na borda de descida do mesmo
if_id_instruc	Contém o valor da instrução recebida do estágio de busca de instruções.
if_id_nextpc	Valor de nextpc recebido do estágio de busca de instruções

id_if_selpcsource	Recebe a saída do comparador caso selbrjumpz (saída do módulo de controle) seja 2'b10, 1 caso selbrjumpz seja 2'b01 e 0 caso contrário
id_if_rega	Recebe o valor de reg_id_ass_dataa
id_if_pcimd2ext	Recebe if_id_nextpc + {{16{if_id_instruc[15]}},if_id_instruc[15:0]}<<2'b10
id_if_pcindex	Recebe {if_id_nextpc[31:28],if_id_instruc[25:0]<<2'b10}
id_if_selpctype	Recebe o valor da saída selpctype do módulo de controle
id_ex_selalushift	Recebe o valor da saída selalushift do módulo de controle
id_ex_selimregb	Recebe o valor da saída selimregb do módulo de controle
id_ex_aluop	Recebe o valor da saída aluop do módulo de controle
id_ex_unsig	Recebe o valor da saída unsig do módulo de controle
id_ex_shiftop	Recebe o valor da saída shiftop do módulo de controle
id_ex_shiftamt	Recebe o valor de reg_id_dataa
id_ex_rega	Recebe o valor de reg_id_dataa
id_ex_readmem	Recebe o valor da saída readmem do módulo de controle
id_ex_writemem	Recebe o valor da saída writemem do módulo de controle
id_ex_regb	Recebe o valor de reg_id_datab
id_ex_imedext	Recebe o valor de if_id_instruc[15:0] e o estende para 32 bits, conservando o sinal
id_ex_selwsource	Recebe o valor da saída selwsource do módulo de controle caso ele tenha sido modificado como descrito abaixo, ou apenas selwsource[0]
id_ex_regdest	Recebe if_id_instruc[15:11] quando selregdest é 1 e if_id_instruc[20:16] quando selregdest 0. Caso o controle não tenha sido modificado, como descrito abaixo, a mesma seleção deve ser feita mas considerando-se apenas selregdest[0]
id_ex_writereg	Recebe o valor da saída writereg do módulo de controle
id_ex_writeov	Recebe o valor da saída writeov do módulo de controle
id_reg_addra	Recebe if_id_instruc[25:21]
id_reg_addrb	Recebe if_id_instruc[20:16]
reg_id_dataa	Valor lido de forma síncrona do banco de registradores na posição especificada por id_reg_addra
reg_id_datab	Valor lido de forma síncrona do banco de

	registradores na posição especificada por id_reg_addrb
reg_id_ass_dataa	Valor lido de forma assíncrona do banco de registradores na posição especificada por id_reg_addra. Deve ser conectado na porta a do comparador
reg_id_ass_datab	Valor lido de forma assíncrona do banco de registradores na posição especificada por id_reg_addrb. Deve ser conectado na porta b do comparador

Execute

O estágio de execução é responsável por receber operandos do estágio de decodificação, assim como a indicação de qual operação será realizada e finalmente realizá-la. O resultado é passado para o estágio de memória.

```
module Execute (
    input                   clock,
    input                   reset,
    //Decode
    input                   id_ex_selalushift,
    input                   id_ex_selimregb,
    input          [2:0]    id_ex_aluop,
    input                   id_ex_unsig,
    input          [1:0]    id_ex_shiftop,
    input          [4:0]    id_ex_shiftamt,
    input          [31:0]   id_ex_rega,
    input                   id_ex_readmem,
    input                   id_ex_writemem,
    input          [31:0]   id_ex_regb,
    input          [31:0]   id_ex_imedext,
    input                   id_ex_selwsource,
    input          [4:0]    id_ex_regdest,
    input                   id_ex_writereg,
    input                   id_ex_writeov,
    //Fetch
    output reg              ex_if_stall,
    //Memory
    output reg              ex_mem_readmem,
    output reg              ex_mem_writemem,
    output reg     [31:0]   ex_mem_regb,
    output reg              ex_mem_selwsource,
    output reg     [4:0]    ex_mem_regdest,
    output reg              ex_mem_writereg,
    output reg     [31:0]   ex_mem_wbvalue
);
```

Sinal	Função
clock	Sinal de clock para controle do acesso. As

	operações no estágio de busca de instruções são realizadas na subida do clock
reset	Sinal de reset assíncrono que coloca todos os registradores deste módulo em 0 na borda de descida do reset
id_ex_selalushift	Sinal que seleciona entre o resultado do *shifter* e da ALU
id_ex_selimregb	Sinal que define o que deve ser conectado à porta b da ALU. Caso seja 1, id_ex_imedext deve ser conectado e, caso contrário, id_ex_regb deve ser conectado
id_ex_aluop	Sinal a ser conectado à porta op da ALU
id_ex_unsig	Sinal a ser conectado à porta unsig da ALU
id_ex_shiftop	Sinal a ser conectado à porta shiftop do shifter
id_ex_shiftamt	Sinal a ser conectado à porta shiftamt do *shifter*
id_ex_rega	Sinal a ser conectado à porta a da ALU
id_ex_readmem	Sinal que indica que a instrução fará uma leitura
id_ex_writemem	Sinal que indica que a instrução fará uma escrita
id_ex_regb	Sinal a ser conectado à porta in do *shifter*
id_ex_imedext	Sinal que contém o valor imediato para instruções que operam imediatos
id_ex_selwsource	Sinal a ser repassado para o estágio de memória
id_ex_regdest	Sinal a ser repassado para o estágio de memória
id_ex_writereg	Sinal que indica que a instrução faz escrita no banco de registradores
id_ex_writeov	Sinal que indica que a instrução faz escrita no banco de registradores mesmo quando acontece um *overflow* na operação realizada pela ALU
ex_if_stall	Indica que um *stall* deve ser inserido, pois a instrução fará um acesso à RAM. Recebe 1 caso id_ex_readmem seja 1 ou id_ex_writemem seja 1
ex_mem_readmem	Recebe id_ex_readmem
ex_mem_writemem	Recebe id_ex_writemem
ex_mem_regb	Recebe id_ex_regb
ex_mem_selwsource	Recebe id_ex_selwsource
ex_mem_regdest	Recebe id_ex_regdest
ex_mem_writereg	Recebe (!aluov | id_ex_writeov) & id_ex_writereg
ex_mem_wbvalue	Recebe a saída do *shifter* caso id_ex_selalushift seja 1 e a saída da ALU caso

	contrário

Memory

O módulo de memória é responsável por ler e escrever dados da/para a RAM. Este estágio tem prioridade máxima para acessar a RAM.

```
module Memory (
    input                   clock,
    input                   reset,
    //Execute
    input                   ex_mem_readmem,
    input                   ex_mem_writemem,
    input         [31:0]    ex_mem_regb,
    input                   ex_mem_selwsource,
    input         [4:0]     ex_mem_regdest,
    input                   ex_mem_writereg,
    input         [31:0]    ex_mem_wbvalue,
    //Memory Controller
    output                  mem_mc_rw,
    output                  mem_mc_en,
    output        [17:0]    mem_mc_addr,
    inout         [31:0]    mem_mc_data,
    //Writeback
    output reg    [4:0]     mem_wb_regdest,
    output reg              mem_wb_writereg,
    output reg    [31:0]    mem_wb_wbvalue
);
```

Sinal	Função
clock	Sinal de clock para controle do acesso. As operações no estágio de busca de instruções são realizadas na subida do clock
reset	Sinal de reset assíncrono que coloca todos os registradores deste módulo em 0 na borda de descida do mesmo
ex_mem_readmem	Sinal que indica se a instrução fará uma leitura
ex_mem_writemem	Sinal que indica se a instrução fará uma escrita
ex_mem_regb	Valor a ser escrito na memória caso a instrução faça uma escrita
ex_mem_selwsource	Sinal que seleciona o que será escrito no banco de registradores, o valor lido da RAM ou o resultado do estágio de execução
ex_mem_regdest	Endereço do registrador a ser escrito
ex_mem_writereg	Sinal que indica se a instrução faz uma escrita no banco de registradores
ex_mem_wbvalue	Resultado do estágio de execução
mem_mc_rw	Recebe (!ex_mem_readmem & ex_mem_writemem), que quando é 1 indica uma escrita
mem_mc_en	Indica que a instrução faz um acesso à RAM e deve ser 1 caso ex_mem_readmem ou

	ex_mem_writemem seja 1
mem_mc_addr	Recebe ex_mem_wbvalue[17:0] indicando o endereço que será lido na RAM
mem_mc_data	Recebe ex_mem_regb quando mem_mc_rw é 1 e recebe um valor vindo do controlador de memória caso contrário
mem_wb_regdest	Recebe o valor de ex_mem_regdest
mem_wb_writereg	Recebe o valor de ex_mem_writereg
mem_wb_wbvalue	Recebe mem_mc_data caso ex_mem_selwsource seja 1 e ex_mem_wbvalue caso contrário

Writeback

A unidade de escrita em registradores é responsável por receber o valor lido ou repassado pelo estágio de memória e escrever no banco de registradores caso a instrução faça escrita.

```
module Writeback (
    //Memory
    input      [4:0]     mem_wb_regdest,
    input                mem_wb_writereg,
    input      [31:0]    mem_wb_wbvalue,
    //Registers
    output               wb_reg_en,
    output     [4:0]     wb_reg_addr,
    output     [31:0]    wb_reg_data
);
```

Sinal	Função
mem_wb_regdest	Endereço que indica qual registrador será escrito
mem_wb_writereg	Sinal que indica se a instrução fará uma escrita
mem_wb_wbvalue	Valor a ser escrito no registrador endereçado por mem_wb_regdest
wb_reg_en	Recebe o valor de mem_wb_writereg
wb_reg_addr	Recebe o valor de mem_wb_regdest
wb_reg_data	Recebe o valor de mem_wb_wbvalue

MemController

O módulo controlador de memória funciona como uma interface de acesso à memória RAM. Tanto os estágios de busca de instruções quanto o estágio de memória fazem acesso à RAM através desse módulo.

```
module MemControler (
    input                clock,
    input                reset,
    //Fetch
    input                if_mc_en,
    input      [17:0]    if_mc_addr,
    output     [31:0]    mc_if_data,
    //Memory
```

```
    input               mem_mc_rw,
    input               mem_mc_en,
    input       [17:0]  mem_mc_addr,
    inout       [31:0]  mem_mc_data,
    //Ram
    output      [17:0]  mc_ram_addr,
    output              mc_ram_wre,
    inout       [15:0]  mc_ram_data
);
```

Sinal	Função
clock	Sinal de clock para controle do acesso. As operações de acesso à RAM são realizadas na subida do clock
reset	Sinal de reset assíncrono que coloca todos os registradores deste módulo em 0 na borda de descida do mesmo
if_mc_en	Indica quando o estágio de busca de instruções, deseja ler uma instrução da RAM. Mesmo que este sinal seja 1, mem_mc_en deve estar em 0 para habilitar o acesso pelo estágio de busca de instruções
if_mc_addr	Endereço referente ao dado acessado pelo estágio de busca de instruções
mc_if_data	Dado completo lido da RAM a ser enviado para a unidade de busca de instruções
mem_mc_rw	Indica quando a instrução corrente no estágio de memória realizará uma escrita na RAM
mem_mc_en	Indica quando a instrução corrente no estágio de memória realizará um acesso à RAM. Este acesso tem prioridade sobre os acessos feitos pelo estágio de busca de instruções
mem_mc_addr	Endereço referente ao dado acessado pelo estágio de memória
mem_mc_data	Dado completo lido da RAM a ser enviado para o estágio de memória ou dado recebido do estágio de memória a ser escrito na RAM
mc_ram_addr	Endereço da RAM no qual o acesso será feito. O endereço utilizado pelos módulos do MIPS assume que cada posição da memória armazena apenas um byte e portanto uma palavra está distribuída em quatro posições, ao invés de duas. Sendo assim, o endereço, addr, recebido deve ser dividido por 2 e então lido, addr/2 (parte mais significativa) e addr/2 + 1 (parte menos significativa)
mc_ram_wre	Indica, quando em 0, que a operação realizada na RAM será de escrita. A operação só é de escrita quando ((!mem_mc_en & if_mc_en) |

	!mem_mc_rw)
mc_ram_data	Metade do dado lido/escrito na RAM a ser enviado/recebido dos estágios de busca de instruções ou de memória

Banco de Registradores

O banco de registradores foi alterado e agora possui duas saídas de dados assíncronas, além das saídas síncronas já implementadas.

```
module Registers (
    input                   clock,
    input                   reset,
    input          [4:0]    addra,
    output reg     [31:0]   dataa,
    output         [31:0]   ass_dataa,
    input          [4:0]    addrb,
    output reg     [31:0]   datab,
    output         [31:0]   ass_datab,
    input                   enc,
    input          [4:0]    addrc,
    input          [31:0]   datac
);
```

Sinal	Função
ass_dataa	Dado A lido do banco de registradores de forma assíncrona da posição especificada pro addra
ass_datab	Dado B lido do banco de registradores de forma assíncrona da posição especificada pro addrb

MIPS

O módulo final do trabalho é o Mips, que instancia todos os módulos referentes aos estágios do pipeline, além do banco de registradores, controlador de memória e cria todas as conexões entre eles. Este módulo também é responsável por gerar o **clock mais lento** necessário nos estágios do *Pipeline*.

A interface do Mips só possui os inputs clock e reset, além dos sinais para acessar a RAM. Essa interface é mostrada a seguir:

```
module Mips (
    input clock,
    input reset,
    //RAM
    output    [17:0]    addr,
    inout     [15:0]    data,
    output              wre,
    output              oute,
    output              hb_mask,
    output              lb_mask,
    output              chip_en
```

```
);

    // ...

endmodule
```

RAM

A memória RAM será responsável por armazenar as instruções a serem executadas. Se conecta diretamente ao MIPS.

```
module Ram (
    input          [17:0]    addr,
    inout          [15:0]    data,
    input                    wre,
    input                    oute,
    input                    hb_mask,
    input                    lb_mask,
    input                    chip_en
```

Sinal	Função
addr	Endereço de acesso à memória RAM
data	Dado a ser escrito/lido da memória
wre	Se 0 indica que a operação na RAM é de leitura
oute	No caso do trabalho deve ser 0.
hb_mask	Higher Byte Mask. Se 1 seta todo o byte superior de data para 1. No caso do trabalho deve ser 0.
lb_mask	Lower Byte Mask. Se 1 seta todo o byte inferior de data para 1. No caso do trabalho deve ser 0.
chip_en	Informa se Ram está ativa. 0 para ativada. No caso do trabalho deve ser 0.

```
);
```

Testes

A entrega consiste basicamente na integração da unidades já desenvolvidas, realizando a implementação do módulo MIPS. Dessa forma o seu maior foco deverá ser nos testes.

Para realizar testes no MIPS completo vocês podem criar um testbench que instancie o MIPS e a RAM e utilizar as funções de sistema do Verilog **$readmemh/$readmemb** que permitem que valores sejam carregados em memórias do circuito, no caso, na RAM fornecida.

Criem códigos assembly baseados no conjunto de instruções do MIPS implementado, convertam-o para hexadecimal ou binário e carreguem-o na RAM para verificar o funcionamento do processador de vocês. Não se esqueçam e incluir os códigos criados para os testes no arquivo enviado.

Lembrem-se de listar quais os sinais utilizaram para verificar o comportamento do processador. Como vocês podem ver agora estamos lidando com muitos sinais.

Vocês podem utilizar o MARS (http://courses.missouristate.edu/kenvollmar/mars/) para gerar o código hexadecimal para vocês, mas enviem o código assembly desenvolvido também. Além disso,

fiquem atentos para as diferenças do MARS com relação MIPS implementados por vocês. Nem todas as instruções disponíveis no MARS foram implementadas. O nosso MIPS não tem repasse, ou seja, temos que escrever o assembly pensando quantos ciclos são necessários para que o resultado de algumas instruções fiquem prontos, o PC é controlado de forma diferente, de forma que instruções de *branch* e *jump* devem ser ajustadas manualmente, entre outros. A tabela com os 24 comandos disponíveis também foi disponibilizada.

Uma boa referência para entender e ajustar o código do MARS é
http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html

Dica1: Por padrão, memórias (arrays de mais de uma dimensão, como os registradores do banco de registradores e as posições de memória da RAM) não são colocadas no arquivo de VCD, mas você pode forçar isso, para, por exemplo, conseguir ver o valor de cada registrador do banco de registradores no VCD. Basta fazer, além do **$dumpvars()** que já coloca todos os outros sinais no VCD, um **$dumpvars(0, Inst0.Inst1.memory[i])**, onde Inst0.Inst1.memory[i] é uma referência hierárquica para a posição de memória i do registrador memory dentro de uma instância Inst1 que existem dentro de uma instância Inst0 que existe no seu testbench

Dica2: Coloquem no começo do código assembly de vocês uma ou duas instruções de NOP para evitar problemas com o reset.

Entrega

A entrega deverá ser feita via moodle até o dia 20/05.

Deve ser entregue um arquivo com nome trabalho2_grupo_<numero_do_grupo>.zip contendo uma pasta chamada trabalho2_grupo_<numero_do_grupo>.

Dentro dessa pasta devem constar três diretórios chamados src, tb e doc, que devem conter, respectivamente, os arquivos referentes aos módulos implementados, os arquivos correspondentes aos testbenches e a documentação sobre cada um dos módulos implementados.

Os arquivos dos módulos implementados devem ter o seu nome e a interfaces dos módulos conforme a especificação. Vale lembrar que Verilog é case sensitive.

Testbenches devem ter o seguinte padrão de nomes: tb_<nome_do_arquivo_sendo_testado>. Por exemplo, o testbench de um arquivo chamado Mips.v deve chamar tb_Mips.v.

A documentação sobre cada um dos módulos deve ser sucinta (menos de uma página deve ser suficiente) e pode ser entregue em qualquer formato, .txt, .pdf, .doc, mas preferencialmente com um nome que indique a qual módulo ela se refere. Esta documentação deve conter uma breve descrição do módulo implementado, eventuais decisões de projeto e o nome das equipes de teste e desenvolvimento.

Além dos itens anteriores, as seguintes perguntas devem ser respondidas:

1. Por que é necessário que o clock do Mips seja mais lento que o clock da RAM?
2. Qual a função do módulo Control.v?
3. Liste pelo menos 3 instruções do MIPS32 que não foram implementadas, informando o tipo e o formato.

4. Qual a necessidade do extensor de sinal no bloco de Decode?
5. Quais seriam as modificações necessárias no MIPS para torná-lo um processador de 64 bits?

Qualquer dúvida ou ambiguidade com relação à especificação, por favor, entrem em contato comigo. Porém estarei disponível somente até o dia **17/05**. Depois disso somente em casos excepcionais responderei aos e-mails. Por isso, comecem o trabalho o quanto antes.

Atrasos sofrerão penalizações e cópias não serão toleradas.