



UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

**ESPECIFICAÇÕES DO PROJETO DE SUBCONJUNTO DE
INSTRUÇÕES DE UM PROCESSADOR MIPS**

IF674 - Infraestrutura de Hardware

MONITORES DE INFRAESTRUTURA DE HARDWARE - EC

RECIFE, ABRIL DE 2017

Professora: Edna Natividade da Silva Barros

Sumário

1	Apresentação	3
2	Especificações da CPU	4
3	Pilha	6
4	Desvios	6
5	Componentes Fornecidos	7
5.1	Memória	7
5.2	Registrador de 32 bits	8
5.3	Banco de Registradores	8
5.4	Registrador de Deslocamento	8
5.5	Unidade Lógica e Aritmética (ALU)	9
6	Tratamento de exceções	9
7	Metodologia de Projeto e Cronograma de Avaliação	11
7.1	Primeira Etapa: Projetando a Busca da Instrução + Pequeno Subconjunto de instruções	11
7.1.1	Descrição da unidade de processamento	12
7.1.2	Descrição da Unidade de Controle	13
7.2	Segunda Etapa: Implementando um conjunto completo de instruções do MIPS	14
7.2.1	Instrução Mul	15
8	Cronograma das apresentações	16
9	Formato do relatório	17
9.1	Descrição das Partes do Relatório	18
9.1.1	Capa	18
9.1.2	Índice	19
9.1.3	Descrição dos módulos	19
9.1.4	Descrição das operações	19
9.1.5	Descrição dos estados de controle	19
9.1.6	Máquina de estados da unidade de processamento	20
9.1.7	Conjunto de simulações	20
10	Anexo 1: Instruções para a configuração e carregamento da memória	21
11	Anexo 2: Criando um arquivo .mif usando o montador	22
12	Links úteis	24

1 Apresentação

Neste documento é apresentada a especificação de projeto da disciplina InfraEstrutura de Hardware. A elaboração de tal documento tem como base a experiência adquirida durante os períodos que os escritores desempenharam suas atividades como monitor.

Este documento é composto por três capítulos. No primeiro é apresentada a especificação e as várias etapas de apresentação do projeto. No segundo é dada uma visão geral da CPU que será implementada. Neste capítulo também está presente o repertório de instruções que o processador desenvolvido deve conter e a descrição dos componentes fornecidos pela equipe da disciplina. No terceiro capítulo é apresentada a especificação do relatório de projeto. Ainda está presente um anexo que ensina como configurar e carregar a memória que irá compor o projeto.

Deve-se ressaltar que é dever de cada equipe ler e entender o que aqui está descrito. Caso existam dúvidas, os monitores deverão ser consultados, pois estes estão aptos e a disposição para esclarecer qualquer dúvida que possa existir durante a execução deste trabalho. É importante destacar que ao final de cada capítulo existem algumas observações que devem ser atentamente seguidas pelas equipes. Caso alguma dessas observações não seja seguida, a nota no projeto será coerente com a falha cometida.

A correção do projeto será totalmente baseada neste documento, portanto a equipe que desenvolver o projeto seguindo as recomendações e exigências aqui contidas e conseguir implementar corretamente todas as instruções do repertório do processador não terá problemas com relação à nota que será atribuída ao seu trabalho. Não serão aceitas reclamações que contrariem as regras estabelecidas neste documento.

É importante que o aluno tenha profissionalismo ao fazer seus trabalhos e respeite as regras que estão definidas. Ressalta-se ainda que os monitores não são autorizados a relevar qualquer descumprimento das regras em qualquer parte do trabalho e, caso isso venha a ocorrer, a nota do trabalho entregue refletirá tal descumprimento. É importante lembrar que as regras servem para facilitar o trabalho de alunos e corretores e, caso elas sejam cumpridas a risca, não haverá problemas com relação à correção.

Bom trabalho!

2 Especificações da CPU

O projeto a ser entregue consiste na implementação de um processador, a qual poderá ser feita com base na implementação descrita no livro texto e na figura abaixo apresentada. O processador deverá incluir três blocos básicos: unidade de processamento, unidade de controle e memória. A seguir é dada uma descrição sucinta do processador a ser projetado.

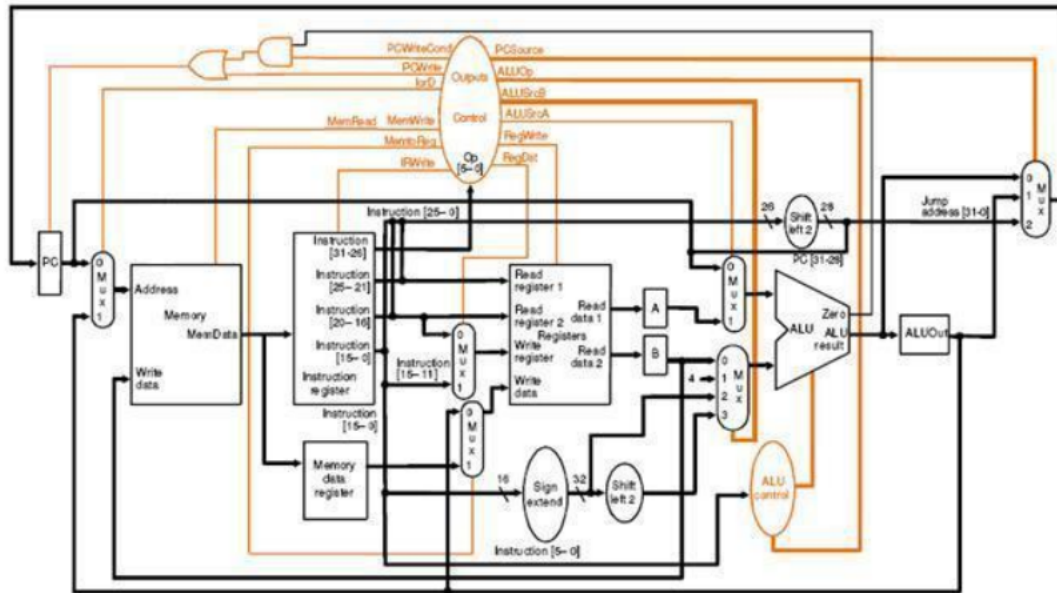


Figura 1 – Circuito base para a construção do projeto.

Inicialmente será projetada a unidade de processamento e a seguir a unidade de controle. A unidade de processamento será desenvolvida a partir da utilização dos seguintes componentes já descritos em VHDL/SystemVerilog e fornecidos através da página da disciplina:

1. ALU
2. Registrador de Deslocamento
3. Registrador de 32 bits
4. Registrador de Instruções
5. Banco de Registradores
6. Memória

Após o projeto da unidade de processamento com a definição de todos os seus componentes e suas respectivas portas de entrada e saída, deverá ser projetada a unidade de controle, que será implementada como uma máquina de estados finita (FSM).

O processador possui 32 registradores de propósito geral, cada um de 32 bits. Um subconjunto das instruções do MIPS deverá ser implementado, de acordo com esta especificação. As instruções estão descritas neste capítulo e agrupadas por formato. Um resumo dos formatos existentes pode ser visto na Tabela 1.

Formato	[31..26]	[25..21]	[20..16]	[15..11]	[10..6]	[5..0]
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	address/immediate		
J	opcode				offset	

Tabela 1 – Formatos de instruções do MIPS

Assembly	opcode	rs	rt	rd	shamt	Funct	Comportamento
add rd, rs, rt	0x0	rs	rt	rd	0x0	0x20	$rd \leftarrow rs + rt$
addu rd, rs, rt	0x0	rs	rt	rd	0x0	0x21	$rd \leftarrow rs + rt *$
and rd, rs, rt	0x0	rs	rt	rd	0x0	0x24	$rd \leftarrow rs \& rt$
jr RS	0x0	rs	-	-	0x0	0x8	$PC \leftarrow rs$
sll rd, rt, shamt	0x0	-	rt	rd	shamt	0x0	$rd \leftarrow rt \ll shamt$
sllv rd, rt, rs	0x0	rs	rt	rd	0x0	0x4	$rd \leftarrow rt \ll rs$
slt rd, rs, rt	0x0	rs	rt	rd	0x0	0x2a	$rd \leftarrow (rs < rt) ? 1 : 0$
sra rd, rt, shamt	0x0	-	rt	rd	shamt	0x3	$rd \leftarrow rt \gg shamt **$
srav rd, rt, rs	0x0	rs	rt	rd	0x0	0x7	$rd \leftarrow rt \gg rs **$
srl rd, rt, shamt	0x0	-	Rt	rd	shamt	0x2	$rd \leftarrow rt \gg shamt$
sub rd, rs, rt	0x0	rs	Rt	rd	0x0	0x22	$rd \leftarrow rs - rt$
subu rd, rs, rt	0x0	rs	Rt	rd	0x0	0x23	$rd \leftarrow rs - rt *$
xor rd, rs, rt	0x0	rs	Rt	rd	0x0	0x26	$rd \leftarrow rs \wedge rt$
break	0x0	-	-	-	0x0	0xd	para a execução do programa
nop	0x0	-	-	-	0x0	0x0	no operation
rte	0x10	-	-	-	0x0	0x10	$PC \leftarrow EPC$ (retorno de exceção)

Tabela 2 – Instruções tipo R

* não geram exceção por overflow, caso o resultado tenha mais que 32 bits

** essas instruções devem estender o sinal (deslocamento aritmético)

Assembly	opcode	rs	rt	address/immediate	comportamento
addi rt, rs, imm	0x8	rs	rt	Imm	$rt \leftarrow rs + imm ***$
addiu rt, rs, imm	0x9	rs	rt	Imm	$rt \leftarrow rs + imm ***$
andi rt, rs, imm	0xc	rs	rt	Imm	$rt \leftarrow rs \& imm ***$
beq rs, rt, offset	0x4	rs	rt	Offset	(ver 'desvios')
bne rs, rt, offset	0x5	rs	rt	Offset	(ver 'desvios')
lbu rt, address (rs)	0x24	rs	rt	Address	$rt \leftarrow \text{byte} [\text{address} + (rs)] ****$
lhu rt, address (rs)	0x25	rs	rt	Address	$rt \leftarrow \text{halfword} [\text{address} + (rs)] ****$
lui rt, imm	0xf	-	rt	Imm	$rt[31..15] \leftarrow imm; rt[15..0] \leftarrow 0$
lw rt, address (rs)	0x23	rs	rt	Address	$rt \leftarrow \text{word} [\text{address} (rs)] ****$
sb rt, address (rs)	0x28	rs	rt	Address	$\text{address} (rs) \leftarrow \text{byte} [rt] ****$
sh rt, address (rs)	0x29	rs	rt	Address	$\text{address} (rs) \leftarrow \text{halfword} [rt] ****$
slti rt, rs, imm	0xa	rs	rt	Imm	$rt \leftarrow (rs < imm) ? 1 : 0 ***$
sw rt, address (rs)	0x2b	rs	rt	Address	$[\text{address} + (rs)] \leftarrow \text{word} [rt] ****$
sxori rt, rs, imm	0xe	rs	rt	Imm	$rt \leftarrow rs \wedge imm ***$

Tabela 3 - Instruções tipo I

*** o valor de 'imm' deve ser estendido para 32 bits, estendendo seu sinal (bit mais significativo da constante).

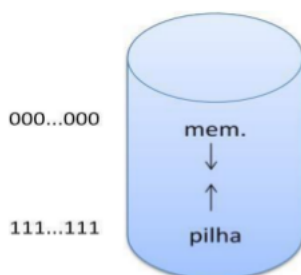
**** o valor de 'address' deverá ser somado ao valor de 'rs' (registrador base) para o cálculo do endereço a ser lido da memória ou do endereço onde os dados devem ser escritos.

assembly	opcode	Offset	comportamento
j	Offset	0x2	(ver 'desvios')
jal	Offset	0x3	(ver 'desvios')

Tabela 4 - Instruções tipo J

3 Pilha

Para permitir a chamada de rotinas reentrantes e recursivas, será possível o armazenamento do registrador 31 numa estrutura do tipo pilha a qual será implementada numa parte da memória como mostrado na figura abaixo:



O endereço do topo da pilha será guardado no registrador 29 (vinte e nove) do banco de registradores, que funciona como o SP (Stack Pointer). Quando o reset é ativado este registrador deverá apontar para o endereço onde começa a pilha (nesta versão: 227). O salvamento e recuperação do registrador 31 na pilha é sempre feito por software (compilador).

4 Desvios

Os desvios do MIPS podem ser classificados como desvios condicionais ou incondicionais. Os desvios condicionais realizam algum teste ou verificação para, somente e não executar o desvio. Os incondicionais sempre executam o desvio. O cálculo do endereço de destino é realizado diferentemente nos dois tipos de desvio:

Desvios condicionais: O valor de 'offset' é multiplicado por 4 e somado ao PC atual. O resultado desta operação é o endereço de destino do salto (desvio), caso o resultado do teste condicional seja verdadeiro.

Exemplo: *beq r3, r4, 0xf* - Supondo que r3 e r4 sejam, ambos, iguais a 0x3a e que o PC atual seja 0x1c, a próxima instrução a ser executada é a que ocupa a posição $(0xf * 4) + 0x1c$, que é a 0x58.

Instruções		
Beq	rs, rt, offset	Realiza o desvio se rs for igual a rt (rs == rt).
Bne	rs, rt, offset	Realiza o desvio se rs for diferente de rt (rs != rt).

Tabela 5 - Desvio condicionais

Desvios incondicionais: No caso de instruções que utilizem offset, este deve ser multiplicado por 4, resultando em um endereço de 28 bits. Como todo endereço deve possuir 32 bits, os 4 bits restantes vêm do PC atual, representando os 4 bits mais significativos do endereço de destino. No caso de instruções que utilizam registrador, o conteúdo do registrador já é o próprio endereço de destino.

Exemplos:

jal 0x1a2b - Supondo que o PC atual seja igual a 0xba12fa00, o endereço de destino será igual a $[(PC \& 0xf0000000) | (0x1a2b * 0x4)]$, que é igual a 0xb00068AC. Note que os 4 bits mais significativos do PC (1011) foram conservados.

jr 6 - Supondo que o r6 seja igual a 0x1a2b, o endereço de destino será igual ao próprio valor de r6, neste caso, 0x1a2b.

Instruções		
j	Offset	Desvia, incondicionalmente, para o endereço calculado.
jal	Offset	Desvia para endereço calculado, porém salva o endereço da instrução subsequente ao jal (endereço de retorno) no registrador r31 (SEMPRE).
jr	Rs	Desvio incondicional para o endereço apontado por rs.

Tabela 6 - Desvio incondicionais

5 Componentes Fornecidos

Nesta seção são descritos os seis componentes do projeto que são fornecidos pela equipe da disciplina. Tais componentes poderão ser baixados a partir da página da disciplina.

5.1 Memória

A memória usada no projeto possuirá palavras de 32 bits com endereçamento por byte. Apesar de o endereço possuir 32 bits, a memória só possui 256 bytes. As entradas e saídas da memória são:



1. **address: entrada de 32 bits**
O endereço de entrada da memória (32 bits).
2. **Data_in: entrada de 32 bits**
A entrada de Dados da memória (32 bits).
3. **clock: entrada de 1 bit**
Bit de clock comum a todo o “computador”.
4. **wr: entrada de 1 bit**
O bit que diz se vai ler ou escrever.
5. **Data_out: saída de 32 bits**
A saída de Dados da memória (32 bits).

Peculiaridades da memória:

1. Enquanto o bit “wr” estiver com o valor 0 (zero) ele estará lendo, quando estiver com o valor 1 (um) ele estará escrevendo.
2. A memória está trigada na subida do clock.
3. Ao se fazer uma requisição de leitura, o valor pedido só estará disponível no segundo pulso de clock após o clock onde foi feita a requisição.
4. A escrita leva apenas um ciclo.
5. Instruções e dados serão armazenados na memória usando a estratégia *big-endian*.

5.2 Registrador de 32 bits

Para armazenar instruções e dados, bem como o endereço de instruções serão utilizados registradores de 32 bits, conforme ilustrado abaixo.

*Obs: o dado que entra em Data_in só passa para Data_out quando o registrador passa 1 ciclo de clock com o sinal **Load** ativo ($Load = 1$).*



5.3 Banco de Registradores

O banco de registradores é composto por 32 registradores de 32 bits cada. Dois registradores podem ser visíveis simultaneamente.

A leitura dos registradores é combinacional, isto é, se os valores nas entradas **ReadRegister1** ou **ReadRegister 2** forem alteradas, os valores nas saídas **ReadData1** ou **ReadData2** podem ser alterados. O registrador a ser escrito é selecionado pela entrada **WriteRegister** e quando a entrada **RegWrite** é ativada (igual a 1) o registrador selecionado recebe o conteúdo da entrada **WriteData**. O sinal de **reset** limpa todos os registradores e é assíncrono.



5.4 Registrador de Deslocamento

O registrador de deslocamento deve ser capaz de deslocar um número inteiro de 32 bits para esquerda e para a direita. No deslocamento a direita o sinal pode ser preservado ou não.

O número de deslocamentos pode variar entre 0 e 32 e é especificado na entrada n (5 bits) do registrador de deslocamento. A funcionalidade desejada do registrador de deslocamento é especificada na entrada shift conforme figura abaixo. As atividades discriminadas na entrada shift são síncronas com o clock (ck) e o reset é assíncrono e ele funciona de forma semelhante como um registrador (demora 1 ciclo para sair o valor certo).

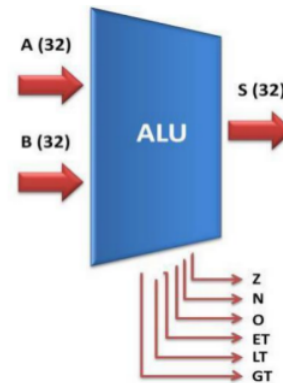
Shift	Descrição
000	Nada a fazer.
001	Load no registrador.
010	Shift a esquerda n vezes.
011	Shift a direita lógico n vezes.
100	Shift a direita aritmético n vezes.
101	Rotação a direita n vezes.
110	Rotação a esquerda n vezes.



5.5 Unidade Lógica e Aritmética (ALU)

A Unidade Lógica e Aritmética (ALU) é um circuito combinacional que permite a operação com números de 32 bits na notação complemento a dois. A funcionalidade é especificada pela entrada F conforme descrito na tabela abaixo.

Função	Operação	Descrição	Flags
000	$S = A$	Carrega A	Z, N
001	$S = A + B$	Soma	Z, N, O
010	$S = A - B$	Subtração	Z, N, O
011	$S = A \text{ and } B$	And lógico	Z
100	$S = A + 1$	Incremento de A	Z, N, O
101	$S = \text{not } A$	Negação de A	Z
110	$S = A \text{ xor } B$	OU exclusivo	Z
111	$S = A \text{ comp } B$	Comparação	EG, GT, LT



6 Tratamento de exceções

O processador deve incluir tratamento de dois tipos de exceções: opcode inexistente e overflow. Na ocorrência de uma exceção, o endereço da instrução que causou a exceção deverá ser salvo no registrador EPC a ser inserido na unidade de processamento e o PC será carregado, posteriormente, com o valor do endereço da rotina de tratamento. O byte menos significativo do endereço da rotina de tratamento está armazenado nos seguintes endereços de memória: 254 (opcode inexistente) e 255 (overflow).

A rotina de tratamento estará armazenada na memória a partir do endereço 228 (decimal) até o endereço 251 (decimal). Esta rotina armazenará o valor 1 no registrador 30 e para a execução de um programa no caso de um opcode inexistente. No caso de um overflow, o valor 2 deverá ser armazenado no registrador 30 e a execução do programa deve continuar.

Basicamente os passos que projeto deve ter implementados após ocorrer uma exceção são os seguintes:

1. Ao ser detectada uma exceção o endereço da instrução que a causou deve ser salvo no registrador EPC.

2. O byte localizado na posição de memória 254 ou 255 (dependendo da exceção) deverá ser lido e estendido para 32 bits. Essa extensão é feita apenas com zeros.
3. O número de 32 bits resultante do passo anterior deverá ser o novo valor de PC. Após completar tais passos a rotina de tratamento de exceção que está armazenada na memória será executada, fazendo assim por software a tarefa de armazenar o valor adequado no registrador 30.

Observação: veja que os passos que acabam de ser descritos não são referentes à instrução *rte*, pois esta instrução deve apenas armazenar em PC o valor contido atualmente em EPC.

Observação 2:

*As instruções **addi**, **addiu**, **lui**, **lw**, **lhu** e **lbu** são de extrema importância para o projeto, pois estas instruções sevem para carregar dados da memória ou da própria instrução para que operações sejam realizadas e valores sejam armazenados nos registradores da CPU. Sem tais instruções não é possível avaliar as outras, pois não será possível carregar valores e operá-los. Portanto, caso algum projeto apresente nenhuma das instruções anteriores funcionando, o mesmo terá consideradas erradas todas as instruções que foram implementadas.*

7 Metodologia de Projeto e Cronograma de Avaliação

Neste capítulo são apresentados os procedimentos que devem ser seguidos para o desenvolvimento das várias etapas do projeto do processador descrito no capítulo anterior desta especificação. Para cada uma das etapas os grupos devem apresentar em papel o projeto da unidade de processamento como diagrama de blocos, o projeto da unidade de controle como diagrama de estados e a implementação em SystemVerilog funcionando. Os diagramas de bloco e de estado devem ser feitos em cartolinas que, preferencialmente, terão a cor branca. Caso exista alguma dúvida as equipes deverão consultar os monitores.

7.1 Primeira Etapa: Projetando a Busca da Instrução + Pequeno Subconjunto de instruções

Nesta etapa os grupos apresentarão a implementação da etapa de busca da instrução na memória e a execução de algumas instruções. Os alunos deverão apresentar a unidade de processamento que faz a busca e a unidade de controle. A implementação deverá ser simulada de forma a ser possível visualizar o valor de PC, a informação lida da memória o registrador de instruções e os estados da máquina de estados e as saídas que explicitam essas informações devem possuir os nomes especificados mais a frente.

Esta etapa corresponde a metade (50%) da nota referente ao projeto. Caso o grupo não consiga implementar a busca, a nota desta etapa será zerada e será fornecido para o grupo um projeto com a busca já implementada, mas ainda será necessário que o grupo implemente todo o repertório de instruções neste projeto.

Os alunos deverão apresentar a unidade de processamento e a unidade de controle que realizam a busca de instruções e a execução das instruções descritas nas tabelas 7, 8 e 9. A implementação em SystemVerilog também deverá ser apresentada e deverá ser simulada de forma a ser possível visualizar o valor de PC, a informação lida e escrita da memória, o registrador de instruções, o conteúdo dos registradores lidos, o conteúdo a ser escrito no registrador e conteúdo do registrador que se localiza na saída da ALU.

	assembly	opcode	rs	Rt	rd	shamt	Funct	Comportamento	
	add	rd, rs, rt	0x0	rs	Rt	rd	0x0	0x20	$rd \leftarrow rs + rt$
	and	rd, rs, rt	0x0	rs	Rt	rd	0x0	0x24	$rd \leftarrow rs \& rt$
	sub	rd, rs, rt	0x0	rs	Rt	rd	0x0	0x22	$rd \leftarrow rs - rt$
	xor	rd, rs, rt	0x0	rs	Rt	rd	0x0	0x26	$rd \leftarrow rs \wedge rt$
	break		0x0	-	-	-	0x0	0xd	para a execução do programa
	nop		0x0	-	-	-	0x0	0x0	no operation

Tabela 7 – Instruções tipo R

Assembly	opcode	rs	rt	address/immediate	comportamento
beq	rs, rt, offset	0x4	rs	rt	Offset (ver 'desvios')
bne	rs, rt, offset	0x5	rs	rt	Offset (ver 'desvios')
lw	rt, address (rs)	0x23	rs	rt	Address $rt \leftarrow \text{memory}[\text{address}+(rs)]$ ****
sw	rt, address (rs)	0x2b	rs	rt	Address $\text{memory}[\text{address}+(rs)] \leftarrow (rt)$
lui	rt, immmed.	0xf	-	rt	Imm $rt[31..16] \leftarrow \text{imm}; rt[15..0] \leftarrow 0$

Tabela 8 – Instruções tipo I

**** o valor de 'address' deverá ser somado ao valor de 'rs' (registrador base) para o cálculo do endereço a ser lido da memória ou do endereço onde os dados devem ser escritos.

	assembly	Opcode	Offset	comportamento
j	offset	0x2	Offset	(ver 'desvios')

Tabela 9 - Instruções tipo J

Obs.: é obrigatória a presença das seguintes saídas na simulação no waveform exatamente com os mesmo nomes citados abaixo, sob a possibilidade de haver penalidades caso não haja esses pinos ou estejam com os nomes diferentes. Podem haver outras saídas, desde que estas estejam presentes.

1. **MemData** ->Mostra o valor na saída da memória;
2. **Address**->Mostra o endereço onde será feita a leitura ou escrita na memória. O pino deve estar após o mux;
3. **WriteDataMem** ->Valor que vai ser escrito na memória;
4. **WriteRegister** ->Mostra o número do registrador onde se vai escrever um dado;
5. **WriteDataReg** ->Mostra o dado que vai ser escrito no registrador;
6. **MDR** ->Mostra o dado presente no Memory Data Register;
7. **Alu** ->Mostra o valor que está saindo da ALU;
8. **AluOut** ->Mostra o valor presente na saída do Reg AluOut;
9. **PC** ->Valor presente no PC.
10. **Reg_Desloc** ->Valor presente no registrador de deslocamento.
11. **wr** ->Sinal que controla se o dado será lido ou escrito na memória.
12. **RegWrite** ->sinal que controla a escrita/leitura no banco de registradores.
13. **IRWrite** ->Sinal que controla a escrita no registrador de instruções.
14. **EPC** ->Valor do registrador EPC.
15. **mul_Module** ->Acompanhamento do módulo de multiplicação.
16. **Estado** ->Número do estado que a Unidade de Controle se encontra.

7.1.1 Descrição da unidade de processamento

Os circuitos que irão compor o projeto devem ser desenhados na cartolina tomando como base a figura 1 do Capítulo 2 desta especificação. A quantidade e quais elementos serão desenhados irá depender da forma como cada grupo decidir implementar o conjunto de instruções que é pedido neste trabalho. Apesar dos grupos terem a liberdade de escolher quantas unidades irão compor o circuito do processador, a arquitetura desenvolvida deve dar sentido a cada um dos componentes que a forma. Além disso, é recomendável que o projeto dê ênfase ao melhor aproveitamento de todos os seus componentes, ou seja, não adianta fazer um circuito com várias unidades que façam pouca ou nenhuma atividade. Deve-se sempre pensar em economia de hardware e rapidez de execução das instruções.

Todas as ligações que forem relevantes ao entendimento do circuito sendo projetado deverão estar presentes no desenho da cartolina, sendo que todos os sinais que partem ou chegam à unidade de controle deverão obrigatoriamente ser apresentados. Devem estar presentes também os sinais que

interligam as demais unidades.

Observação: não é necessário desenhar os sinais de clock e reset que estarão presentes na maioria das unidades, pois a presença destes sinais é intuitivamente percebida. Os sinais que saem da unidade de controle deverão estar nomeados, pois isso facilita no entendimento geral do circuito.

7.1.2 Descrição da Unidade de Controle

As equipes deverão apresentar o diagrama de estados da máquina de estados da unidade de controle do processador. Tal diagrama deverá ser desenhado em uma nova cartolina e terá que conter os estados que a unidade de controle está enquanto o processador executa suas instruções. Cada estado deverá conter os valores e os nomes dos sinais de saída da unidade de controle. Não é necessário especificar todos os sinais da unidade em cada estado, porém é necessária a presença de todos os sinais que foram alterados em cada estado específico.

7.2 Segunda Etapa: Implementando um conjunto completo de instruções do MIPS

Nesta etapa deverá ser apresentada a busca e execução de conjunto de instruções do MIPS descrito no capítulo anterior com tratamento de exceções de opcode inexistente, overflow e multiplicação. Os alunos deverão apresentar a unidade de processamento e a unidade de controle que realizam a busca de instruções e que realizam as instruções do subconjunto 1 (etapas anteriores) e a execução do restante das instruções descritas nas tabelas 10, 11 e 12, ou seja, todas as instruções presentes nas tabelas 2, 3 e 4.

Obs: será implementada também a função *Mul*, função que será descrita melhor a seguir.

Assembly	opcode	rs	rt	rd	shamt	Funct	Comportamento
addu rd, rs, rt	0x0	rs	rt	rd	0x0	0x21	$rd \leftarrow rs + rt$ *
jr RS	0x0	rs	-	-	0x0	0x8	$PC \leftarrow rs$
sll rd, rt, shamt	0x0	-	rt	rd	shamt	0x0	$rd \leftarrow rt \ll shamt$
sllv rd, rt, rs	0x0	rs	rt	rd	0x0	0x4	$rd \leftarrow rt \ll rs$
slt rd, rs, rt	0x0	rs	rt	rd	0x0	0x2a	$rd \leftarrow (rs < rt) ? 1 : 0$
sra rd, rt, shamt	0x0	-	rt	rd	shamt	0x3	$rd \leftarrow rt \gg shamt$ **
srav rd, rt, rs	0x0	rs	rt	rd	0x0	0x7	$rd \leftarrow rt \gg rs$ **
srl rd, rt, shamt	0x0	-	Rt	rd	shamt	0x2	$rd \leftarrow rt \gg shamt$
subu rd, rs, rt	0x0	rs	Rt	rd	0x0	0x23	$rd \leftarrow rs - rt$ *
rte	0x10	-	-	-	0x0	0x10	$PC \leftarrow EPC$ (retorno de exceção)

Tabela 10 – Instruções tipo R

* não geram exceção por overflow, caso o resultado tenha mais que 32 bits

** essas instruções devem estender o sinal (deslocamento aritmético)

Assembly	opcode	rs	rt	address/immediate	comportamento
addi rt, rs, imm	0x8	rs	rt	Imm	$rt \leftarrow rs + imm$ ***
addiu rt, rs, imm	0x9	rs	rt	Imm	$rt \leftarrow rs + imm$ ***
andi rt, rs, imm	0xc	rs	rt	Imm	$rt \leftarrow rs \& imm$ ***
lbu rt, address (rs)	0x24	rs	rt	Address	$rt \leftarrow \text{byte} [\text{address} + (rs)]$ ****
lhu rt, address (rs)	0x25	rs	rt	Address	$rt \leftarrow \text{halfword} [\text{address} + (rs)]$ ****
sb rt, address (rs)	0x28	rs	rt	Address	$\text{address} (rs) \leftarrow \text{byte} [rt]$ ****
sh rt, address (rs)	0x29	rs	rt	Address	$\text{address} (rs) \leftarrow \text{halfword} [rt]$ ****
slti rt, rs, imm	0xa	rs	rt	Imm	$rt \leftarrow (rs < imm) ? 1 : 0$ ***
sxori rt, rs, imm	0xe	rs	rt	Imm	$rt \leftarrow rs \wedge imm$ ***

Tabela 11 – Instruções tipo I

*** o valor de ‘imm’ deve ser estendido para 32 bits, estendendo seu sinal (bit mais significativo da constante).

**** o valor de ‘address’ deverá ser somado ao valor de ‘rs’ (registrador base) para o cálculo do endereço a ser lido da memória ou do endereço onde os dados devem ser escritos.

assembly	opcode	Offset	comportamento
jal Offset	0x3	Offset	(ver ‘desvios’)

Tabela 12 - Instruções tipo J

7.2.1 Instrução Mul

Nesta seção serão descritos o funcionamento e a implementação da função Mul.

Observações:

Durante o desenvolvimento destes trabalhos os alunos terão algumas aulas de laboratório para o acompanhamento de seus projetos. Caso ainda existam dúvidas, os alunos poderão procurar os monitores para que sejam feitos os devidos esclarecimentos.

Após o término de cada etapa do projeto os alunos deverão guardar as suas descrições da arquitetura, pois elas serão muito úteis no desenvolvimento da próxima etapa. Recomendamos que os alunos marquem previamente um horário para o esclarecimento de suas dúvidas mandando um e-mail para o monitor responsável por sua equipe.

É obrigatória a presença das saídas presentes no tópico 7.1 na simulação no waveform exatamente com os mesmo nomes citados abaixo, sob a possibilidade de haver penalidades caso não haja esses pinos ou estejam com os nomes diferentes. Podem haver outras saídas, desde que estas estejam presentes.

8 Cronograma das apresentações

Acompanhamento de projeto: 11/04/17, terça, 10:00 às 12:00 no Laboratório de Hardware.

Acompanhamento de projeto: 17/04/17, segunda, 17:00 às 19:00 no Laboratório de Hardware.

Etapa 1: 18/04/17, terça, 10:00 às 13:00 no Laboratório de Hardware.

Acompanhamento de projeto: 27/04/17, quinta, 8:00 às 10:00 no Laboratório de Hardware.

Acompanhamento de projeto: 08/05/17, segunda, 17:00 às 19:00 no Laboratório de Hardware.

Etapa 2: 09/05/17, terça, 10:00 às 13:00 no Laboratório de Hardware.

9 Formato do relatório

O relatório deverá conter as seguintes partes:

1. Capa;
2. Índice;
3. Descrição dos módulos;
4. Descrição das operações;
5. Descrição dos estados de controle;
6. Máquina de estados da unidade de processamento;
7. Conjunto de simulações.

É obrigatória a presença de todas as partes citadas. Uma descrição de cada uma delas é feita na seção seguinte.

9.1 Descrição das Partes do Relatório

9.1.1 Capa

A capa deve conter um cabeçalho com o nome da universidade e do centro onde a disciplina é ensinada. Deve também apresentar o título: Relatório de Projeto (título único que deve estar presente em todos os relatórios). O nome dos alunos que compõem a equipe deve vir logo em seguida, juntamente com seus logins, e por último deve vir a data da entrega do projeto.

Não será necessária nenhuma imagem na capa que tenha como simples objetivo embelezar o trabalho. A capa padrão é definida no molde já descrito, portanto não faça além do que foi pedido.

Ex.:



RELATÓRIO DE PROJETO

EDUARDO BARRETO BRITO, ebb2
LUCAS PONTES DE ALBUQUERQUE, lpa
MIGUEL LUIZ PESSOA DA CRUZ SILVA, mlpcs

RECIFE, 4 DE NOVEMBRO DE 2016
Professora: Edna Natividade da Silva Barros

9.1.2 Índice

Um índice simples deve compor o relatório para que a correção seja facilitada.

9.1.3 Descrição dos módulos

Cada dos módulos que forem implementados* pelas equipes deve ser descrito em detalhes. O objetivo almejado ao construir tais módulos deve estar claramente apresentado, bem como sua funcionalidade dentro do escopo do projeto. Os sinais de entrada e saída devem ser especificados assim como o conjunto de atividades executadas pelo algoritmo interno de cada um para prover os resultados esperados.

Exemplo: Usamos como exemplo a descrição do registrador de 32 bits que é fornecido juntamente à especificação do projeto.

Módulo: Registrador de 32 bits.

Entradas:

Clk (1 bit): representa o clock do sistema.

Reset (1 bit): sinal que, quando ativado zera o conteúdo do registrador.

Load (1 bit): sinal que ativa o carregamento de um valor de entrada no registrador. Entrada (32 bits): vetor de bits que será armazenado no registrador.

Saída (32 bits): vetor de bits que contém o valor armazenado no registrador.

Objetivo:

Módulo criado para armazenamento temporário de valores, que na implementação multiciclo devem ser preservados durante a execução de cada estágio das instruções.

Algoritmo:

Quando valor de Load está ativado e ocorre a ativação do sinal Clk o vetor de bits Entrada é disponibilizado como o sinal de Saída até que o sinal de load esteja novamente ativado na subida de Clk. Caso o sinal Clear esteja ativado o valor da saída passa a ser o vetor zero.

*Multiplexadores não devem ser descritos, pois seu funcionamento é trivial

9.1.4 Descrição das operações

Nesta parte deverão ficar todos os passos que compõem a execução de cada uma das instruções exigidas no projeto destacando todas os módulos envolvidas na sua execução. Cada instrução deverá ser descrita separadamente.

Exemplo:

Instrução add reg1, reg2, reg3

Após identificar a instrução add no estágio de decodificação os valores dos registradores r2 e r3 são carregados a partir do banco de registradores e a operação de soma é realizada na ULA de acordo com o sinal de controle que vem da unidade de controle, posteriormente o resultado é gravado no registrador de destino r1 e uma nova instrução é buscada na memória.

9.1.5 Descrição dos estados de controle

O objetivo aqui é que as equipes apresentem a descrição de cada um dos estados do controle do processador desenvolvido. Não será necessário que o aluno especifique o valor que cada sinal deve receber dentro do estado, pois isso já estará claro no código do projeto, porém é imprescindível que seja descrito o que cada estado deve fazer, ou seja, o que a equipe objetivava ao construir cada um. Deverá ser apresentado também o diagrama de estados do controle em forma de figura que obrigatoriamente deve estar dentro desta seção do relatório (Atenção: não será aceita a visualização do diagrama de

estados gerada pelo software Quartus. Cada equipe deve se dar ao trabalho de gerar uma figura que **represente de forma clara** o que foi pedido).

Exemplo:

Estado: Decodifica

Neste estado o opcode e o function (quando necessário) são analisados. Isso define qual o novo estado que a máquina deve assumir para continuar o processamento.

9.1.6 Máquina de estados da unidade de processamento

Neste segmento do relatório, deve estar presente a imagem da máquina de estados da unidade de processamento, além de qual o número de cada estado (que deve estar explícito no waveform). É proibido usar a imagem gerada automaticamente pelo Software Quartus.

9.1.7 Conjunto de simulações

Por fim, será disponibilizado, ao longo do projeto, alguns arquivos .mif para que os alunos simulem ele em seu projeto e seja escrito, no relatório, valores requisitados pelo monitor. Esta parte terá um modelo que será disponibilizado pela equipe de monitoria.

Observações:

As regras descritas nesse capítulo devem ser seguidas a risca na elaboração do relatório, sendo que reclamações que não estejam de acordo com o que foi exigido nessa especificação não serão consideradas válidas.

Os monitores serão extremamente rígidos com relação a tais regras, portanto os alunos devem ler atentamente a especificação, entender o que foi pedido para, posteriormente, fazer o relatório.

O relatório pode ser entregue impresso e encadernado ou em formato PDF (digital) na página do Google Drive criado pelo monitor e designado para seu grupo.

O relatório do grupo deve ser entregue até a data especificada neste documento, então não serão aceitos documentos atrasados, fazendo com que o grupo leve uma penalidade na nota.

10 Anexo 1: Instruções para a configuração e carregamento da memória

A memória, usada no projeto e disponibilizada nos componentes, usa a estratégia de armazenamento *Big-endian*, que é a estratégia usada pelos processadores Mips.

Exemplo de Big-Endian: Armazenar a palavra 90AB12CD16 (base hexadecimal).

Na implementação Big-Endian, o byte mais significativo é armazenado nos menores endereços da memória, 90AB12CD16 em *Big-endian* escrito na memória ficaria assim:

Address	Value
1000	90
1001	AB
1002	12
1003	CD

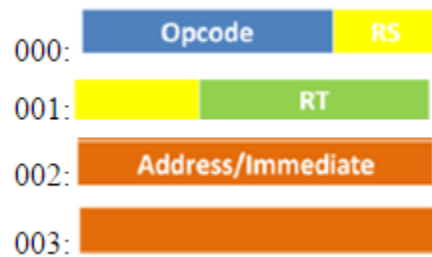
Enquanto na estratégia Little-Endian para a mesma palavra, seria:

Address	Value
1000	CD
1001	12
1002	AB
1003	90

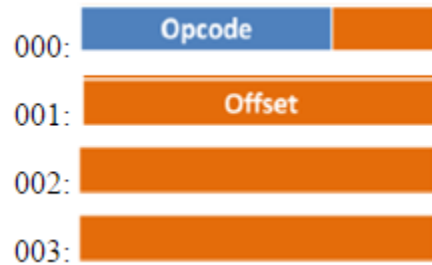
Para o teste do projeto o arquivo .mif deverá ser configurado com um conjunto de instruções específico. Isso é feito com a inserção das instruções nas posições de memória que estão marcadas na margem esquerda de cada linha do arquivo .mif. As figuras abaixo mostram como dispor as instruções de cada tipo nas posições de memória.



Instruções tipo R



Instruções tipo I



Instruções tipo J

Na posição do arquivo Memoria.vhd destacada abaixo insira o nome do arquivo .mif para ele passar a ser parte do seu projeto, onde por padrão está configurado com o nome instrucoes.mif.

```
package ram_constants is
constant DATA_WIDTH : INTEGER := 8;
constant ADDR_WIDTH : INTEGER := 8;
constant INIT_FILE : STRING := "instrucoes.mif";
end ram_constants;
```

Faça a síntese de seu projeto e os testes. Caso seja necessário mudar o arquivo .mif, uma nova Netlist(simulação) deverá ser gerada.

11 Anexo 2: Criando um arquivo .mif usando o montador

É possível criar um arquivo.mif com instruções usando o montador cedido pela monitoria e que está no site da disciplina.

Regras do montador:

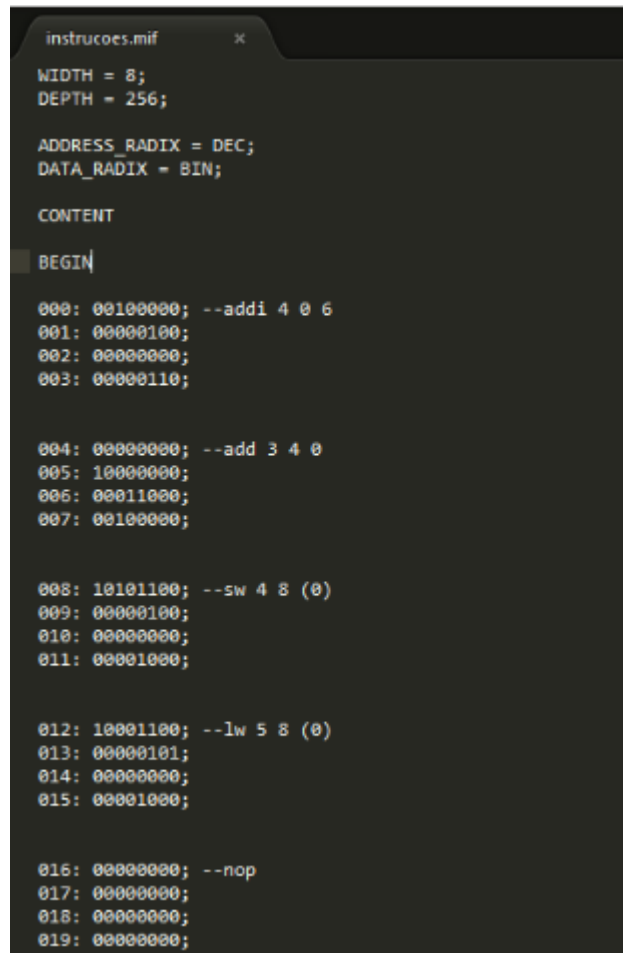
1. No arquivo de nome Assembly.txt devem ser colocadas as instruções que se quer simular no projeto, onde a partir destas instruções será gerado um arquivo .mif.
2. **A primeira linha deve conter a quantidade de instruções que você irá colocar**, seguido dessas instruções, que devem ser no máximo 57.
3. No arquivo .mif as posições a partir da 228 contém as rotinas de tratamento das exceções que não devem ser alteradas mas implementadas conforme a especificação.
4. **Não deve ser usado \$** para expressar o número de um registrador e **nem ()** nas instruções de Load e Store.
5. Nas instruções de Desvio condicional, desvio incondicional, loads e stores, os endereços não devem ser labels (Ex.: beq \$3,\$5, a), mas sim endereços físicos (Ex.: beq 3 5 a).

6. Nas instruções de desvios condicionais e incondicionais deve ser levado em conta que o endereço ainda será multiplicado por 4, o que faz parte do cálculo de endereço.
7. Nas instruções de Load/Store o cálculo do endereço é o mostrado na especificação onde o endereço final é o valor do **registrador base multiplicado por 4 e somado ao deslocamento**.

Exemplo de entrada válida:

```
4
addi 4 0 6
add 3 4 0
sw 4 8 0
lw 5 8 0
```

Saída:



```
instrucoes.mif
WIDTH = 8;
DEPTH = 256;

ADDRESS_RADIX = DEC;
DATA_RADIX = BIN;

CONTENT
BEGIN

000: 00100000; --addi 4 0 6
001: 00000100;
002: 00000000;
003: 00000110;

004: 00000000; --add 3 4 0
005: 10000000;
006: 00011000;
007: 00100000;

008: 10101100; --sw 4 8 (0)
009: 00000100;
010: 00000000;
011: 00001000;

012: 10001100; --lw 5 8 (0)
013: 00000101;
014: 00000000;
015: 00001000;

016: 00000000; --nop
017: 00000000;
018: 00000000;
019: 00000000;
```

O montador não tem um console com resultados dos erros nem acertos. Se o arquivo Assembly.txt estiver correto e você executar o montador, ele vai gerar um arquivo instrucoes.mif e, caso haja algum erro, o arquivo instrucoes.mif não será gerado e você deve consertar o erro e executar de novo o montador.

O arquivo instrucoes.mif deve ser copiado e colocado na pasta onde está a memoria.vhd, além de ser importado para o projeto no Quartus, para então o projeto ser compilado e simulado.

A monitoria irá disponibilizar alguns arquivos de memória .mif com as instruções que estão neles detalhadas para que o projeto seja testado por vocês na confecção do mesmo.

12 Links úteis

[Montador](#);
[Componentes do projeto](#);
[Aula 1 do Quartus \(SD\)](#);
[Aula 2 do Quartus \(SD\)](#);
[Divisão de grupos por monitor](#);
[Cronograma completo da cadeira](#).

Boa sorte e bom trabalho!