Autores:

- Jhone Darts
- Leandro Campos
- Yuri Martins

A Linguagem do Computador

Sistemas Digitais

Universidade Estadual de Feira de Santana

Versão 1.0

Histórico de Revisões

Date	Descrição	Autor(s)
14/08/2016	Inserção de tópicos a serem relatados	Yuri da Silva Martins
15/08/2016	• Adcionando mais conteudos ao relatório • Revisão de todo o conteudo do relatório	Jhone Mendes, Leandro Campos e Yuri Martins

SUMÁRIO

1 | Introdução

1. Propósito do Documento

Este documento da disciplina MI Sistemas Digitais, no qual, tem como objetivo desvendar de que maneira, foi desenvolvido um montador e um simulador. Mostrando como é composta suas arquiteturas, quais são as instruções e seus tipos, apresentando, de que modo, é possível acionar o funcionamento de ambos componentes e sinalizando suas respectivas saídas.

2. Organização Geral do Documento

O presente documento é apresentado como segue:

- **Tópico 2** – Este capítulo apresenta uma visão geral da arquitetura, indicando como está organizada as palavras, como estão expostas as instruções e explicitando a arquitetura geral utilizada;
- **Tópico 3** – Este tópico descreve todos os tipos de intruções aceitas pelo processador;
- **Tópico 4** – Este tópico mostra toda a descrição do montador, contendo sua função e evidenciando sua entrada e sáida de dados do mesmo;
- **Tópico 5** – Este tópico apresenta uma visão geral da arquitetura, com foco em entrada e saída do sistema e arquitetura geral do mesmo;

3. Acrônimos e Abreviações

Sigla	Descrição
RISC	Arquitetura Composta de um numero menor de instruções.
ISA	Camada entre o software e hardware, possuindo um grupo de instruções.
MIPS	Tipo de arquitetura utilizada para o desenvolvimento do montador e simulador.

2 | Visão geral da arquitetura

Utilizamos neste processador um conjunto de 32 registradores, em que estão organizado em uma estrutura de armazenamento destes registradores, contendo em sua arquitetura uma palavra com extensão de 32 bits. A camada ISA foi desenvolvida, utilizando-se do conceitos da Arquitetura RISC e a arquiteturas do MIPS. A seguir, descreveremos a visão geral da arquitetura de alguns aspectos utilizados em nosso processador.

1. Arquitetura 32 bits

Na arquitetura utilizada, uma palavra contém uma extensão de 32 bits, ou seja, 000000 00000 00000 00000 00000 000000, em que cada subconjunto de bits, representa o tipo de instrução na qual foi processada.

2. Arquitetura com 32 registradores

Na arquitetura MIPS na qual foi utilizada, contém um número máximo de 32 registradores, em que estes tais registros são armazenados valores obtidos através de uma determinada instrução. Esses registradores possuem uma organização, como por exemplo de $S0a$S7 são mapeados de 16 até 23.

3. Divisão da palavra 4 bytes

Uma palavra em um registrador é separada com o tamanho de um byte, como a largura da mesma contém 32 bits, acontece uma divisão em 4, com isto, existe a possibilidade de armazenar todos os bits desta palavra.

4. Risc

Um conjunto de instruções simples e com um tamanho reduzido, em que são as mais utilizadas. Esta arquitetura foi empregada neste processador, pela característica do mesmo de ser simples, visto que, a leitura das instruções é feita sequencialmente e uma por vez.

5. MIPS

Neste tipo de arquitetura, onde é definido o conjunto de instruções que está disposto para o desenvolvimento de um processador, contém também a sua organização da estrutura de armazenamento das informações, no qual, é feito por meio de registradores e memória e especificações como deve ser utilizado os seu registradores.

6. 64 instruções

Reunimos 64 tipos de instruções, no qual está dentro do padrão do MIPS, em que o nosso processador é capaz de realizar uma leitura e consequentemente uma tradução.

7. Endereçamento

Com a finalidade de obter um armazenamento, pode ser feito em três formas, tal como, via registrador, imediato e indexado. O endereçamento através do registrador, utiliza apenas os mesmos para realizar as operações, sendo que, os dados já estão armazenados neles. Imediato, é inserido na própria instrução, um valor constante, já estabelecido. Utiliza o endereço da memória, no qual pode ser a base de um vetor e um offset, em que, estabelece o novo endereço que deseja alcançar.

8. Tipos de instrução

Em nosso processador existe grupos de instruções distintas, na quais são elas, aritméticas com cálculos de somar; subtrair; multiplicar; etc.; contem também as instruções de acesso a memória, que são feitas por meio do comando load e store, existindo instruções com característica de desvios e saltos são encontradas com esta natureza jump e por fim, as lógicas possuindo tipos, tal como, and, or e entre outras.

9. Instruções aritméticas

Possuem três operandos, em que, eles são o registrador destino, no qual possui o resultado da operação, um registrador fonte contendo um operando e um outro operando, podendo ser um registrador fonte, ou temporário e até mesmo ser um valor de uma constante.

10. Overflow

Em operações aritméticas, em nosso processador sinalizamos quando acontece um resultado inesperado, nestes tipos de operações, por meio de uma flag. Este overflow acontece quando em um determinado cálculo, excede a quantidade de bits disponível para sua representação.

11. Organização da memória

A memória, é compartilhada entre instruções e dados, sua organização está disposta da seguinte maneira, as instruções antecedem as informações dos dados, ou seja, logo após a leitura das instruções, são obtidos os dados.

3 | Particularidades da Arquitetura e conjunto de instruções

1. Conjunto de Instruções

No nosso processador projetado como já foi dito, atendem a um conjunto de 64 instruções quem podem ser lidas e traduzidas, cada uma totalizando 32 bits de extensão, das quais estão agrupadas por: Aritmética, lógica, Armazenamentos e carregamento de dados e saltos.

1.1. Aritméticas

Soma, multiplica, subtrai, dividi e incrementos.

	Código de operação	Instrução	Operação
Operações aritméticas			
	000000	ADD $d, $s, $t	$d = $s + $t
	001000	ADDI $d, $s, c	$d = $s + c
	001001	ADDIU $d, $s, c	$d = $s + c
	000000	ADDU $d, $s, $t	$d = $s + $t
	011100	CLZ $d, $s	d = COUNTLEADINGZEROS($
	011100	CLO $d, $s	d = COUNTLEADINGONES($s
	001111	LUI $d, c	R[$d] ← c ‖ 0^16
	000000	SUB $d, $s, $t	$d = $s - $t
	000000	SUBU $d, $s, $t	$d = $s - $t
	011111	SEB $d, $t	$d = SignExtend($t)
	011111	SEH $d, $t	$d = SignExtend($t)

Figura 3.1: **Operações Aritméticas**

1.2. Lógicas

Operações lógicas			
	000000	AND $d, $s, $t	$d = $s & $t
	001100	ANDI $d, $s, c	$d = $s & c
	000000	NOR $d, $s, $t	$d = ~($s | $t)
	011111	WSBH $d, $t	$d = WSBH($t)
	000000	OR $d, $s, $t	$d = $s | $t
	001101	ORI $d, $s, c	$d = $s | c
	000000	XOR $d, $s, $t	$d = $s ^ $t
	001110	XORI $d, $s, c	$d = $s ^ c
	011111	EXT $t, $s, pos, size	$rt = EXT($s, size, pos)
	011111	INS rt, rs, pos, size	$rt= InsertField($rt, $rs, msb, lsb)

Figura 3.2: **Operações Lógicas**

1.3. Load/Store

Operações de Load e Store			
	100000	LB $d, c($s)	$d = offset(base)
	100011	LW $d, c($s)	$d = offset(base)
	100001	LH $d, c($s)	$d = offset(base)
	101000	SB $d, c($s)	memory[base+offset] = $d
	101001	SH $d, c($s)	memory[base+offset] = $d
	101011	SW $d, c($s)	memory[base+offset] = $d

Figura 3.3: **Operações de Load e Store**

1.4. Deslocamento

Operações de Deslocamento			
	000000	SLL $d, $s, c	$d = $s << c
	000000	SLLV $d, $s, $t	$d = $s<<$t
	000000	SRL $d, $s, c	$d = $s >> c
	000000	SRA $d, $s, c	$d = $s >> c (arithmetic)
	000000	SRAV $d, $s, $t	$d = $t >> $s] (arithmetic)
	000000	SRLV $d, $s, $t	$d = $t >> $s
	000000	ROTR $d, $s, c	$d = $s :: sa
	000000	ROTRV $d, $s, $t	$d = $s :: $t

Figura 3.4: **Operações de Deslocamento**

1.5. Saltos

Saltos e Branches			
	000100	BEQ $s, $t, L	if $s == $t, go to L
	000111	BGTZ $s, offset	if $s>0, pule
	000101	BNE $s, $t, L	if $s != $t, go to L
	000001	BLTZ $s,offset	if GPR[rs] < 0, pule
	000010	J L	go to L
	000000	JR $ra	go to $ra(address)
	000011	JAL L	$ra = PC + 4, go to L
	000000	JALR $d, $s	$d = addres_return ; pc = $s

Figura 3.5: **Operações de Saltos e Branches**

1.6. Testes Condicionais e troca

Testes condicionais e operações de troca			
	000000	SLT $d, $s, $t	if $s < $t, $d = 1 (else $d = 0)
	001010	SLTI $d, $s, c	if $s < c, $d = 1 (else $d = 0)
	001011	SLTIU $d, $s, c	if $s < c, $d = 1 (else $d = 0)
	000000	SLTU $d, $s, $t	if $s < $t, $d = 1 (else $d = 0)
	000000	MOVN $d, $s, $t	if $t != 0, $d = $s
	000000	MOVZ $d, $s, $t	if $t == 0, $d = $s

Figura 3.6: **Operações de Testes Condicionais e Troca**

1.7. Acesso ao acumulador

Operações de acesso ao acumulador			
	000000	MFHI $d	$d = HI
	000000	MFLO $d	$d = LO
	000000	MTHI $s	HI = $s
	000000	MTLO $s	LO = $s

Figura 3.7: **Operações de Acesso ao Acumulador**

4 | Características do montador e seu funcionamento

0.1. Arquivo de entrada

O arquivo de entrada é um arquivo de texto no formato .asm, possuindo uma sintaxe em assembly. Este arquivo deve possuir a seguinte estrutura: As palavras iniciadas em

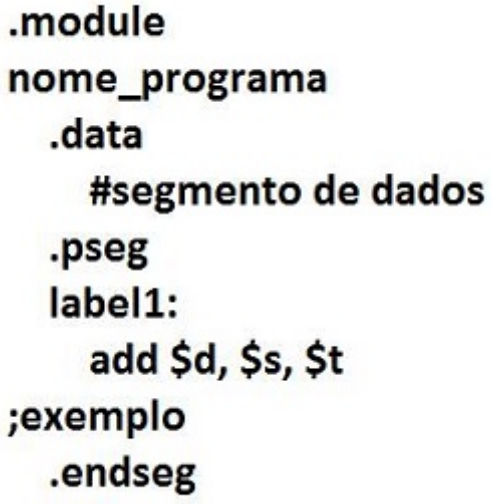

Figura 4.1: **Estrutura do código de entrada**

ponto(.) são as diretivas que dão nome a parte do código. O .module indica o nome do programa. O .data indica o inicio do segmento de dados. O .pseg, o fim do segmento de dados e inicio do segmento de código e o endseg fecha este bloco. Dentro do segmento de dados as palavras são especificadas por tipo, neste caso apenas o tipo .word é aceito, seguido do valor. No segmento de código é onde serão escritas as instruções que o programa irá executar. São aceitas além das 64 instruções descritas neste documento, as instruções li e move, que representam o carregamento de um valor imediato em uma variavel e o carregamento de um valor de um registrador para outro respectivamente, porem não implementam funçoes novas, li é a função ori e move é a função addi ambas com o valor do imediato igual a zero.

1. Função

O montador tem a função de converter o codigo escrito em assembly para linguagem de maquina, uma representação hexadecimal.

2. Fluxograma

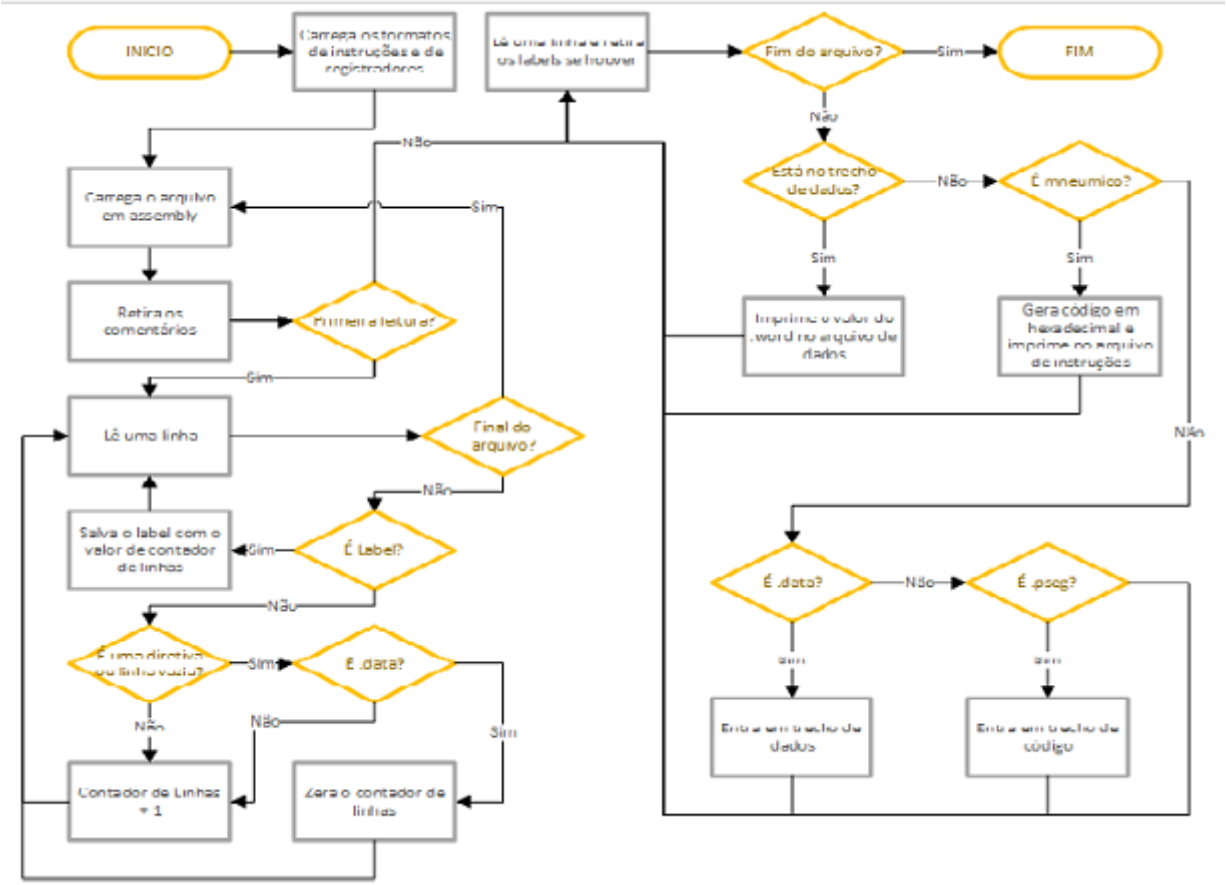

Figura 4.2: **Fluxograma do Montador**

3. Arquivo de saída

Após a execução o montador gera dois arquivos com o mesmo nome do arquivo de entrada seguidos por $_{d}ata$", $paraoarquivodedados, ou$"$_{p}seg$", $paraoarquivoquecontemasinstrues. Ambosarquivosaescritae$

4. Como usar

Execute o projeto apartir de uma IDE de programação, sugiro NetBeansIDE 8.01 usada no desenvolvimento. O consoleirá exibir um menu, onde pode-se escolher um dos codigos de testes ou digitar o nome de um outro. Após a escolha será feita a tradução do código, se tudo ocorrer bem será exibida a mensagem "Tradução feita com sucesso!", caso contrario "Houston, temos um problema". A falta de sucesso na tradução se dá apenas por uma má escrita do codigo. Os arquivos de entrada devem estar em na pasta "codes", que será a pasta usada pra salvar os arquivos de saída.

5 | Características do simulador e seu funcionamento

1. Arquivo de entrada

Os arquivso de entrada são gerados pelo montador no formato .txt, no qual, são criados em após sua executação traduzindo algum código assembly, contendo as palavras representadas em hexadecimais. São dois arquivos, um que contem os dados que ocuparão a memória de dados, e o segundo que ocupará a memoria de instruções.

1.1. Função

Simular o comportamento de determinadas funções em um processador que siga as especificações deste documento. Esta simulação pode ser notada pela exibição dos valores dos registradores e da memória de dados.

2. Arquivo de saída