Universidade de Brasília

Instituto de Ciências Exatas

Departamento de Ciência da Computação

116394 - Organização e Arquitetura de Computadores

Relatório - Trabalho II: Simulador MIPS

Aluno:
Lucas Santos - 14/0151010

Professor:
Ricardo Jacobi

18 de setembro de 2016

1 Descrição do Problema

O trabalho consiste na elaboração de um simulador da arquitetura MIPS, como o *MARS*, em uma linguagem de alto nivel. O simulador deve implementar as seguintes funções: *fetch()*, busca da instrução; *decode()*, decodificação da instrução; *execute()*, execução da instrução. O programa binário a ser executado deve ser gerado a partir do *MIPS*. O simulador implementado deve ler os arquivos binarios contendo as instruções e os dados para sua memória e executá-lo.

As instruções deste trabalho começam no endereço $0x00000000$ e se encerram no endereço $0x00000044$, enquanto os dados no endereço $0x00002000$ e acabam no endereço $0x0000204c$. A memória simulada é definida com o tamanho de $4K$Words, ou seja, $16K$Bytes. Tomando como exemplo, o código a seguir:

```
1  .data
2
3  primos: .word   1,3,5,7,11,13,17,19
4  size:   .word   8
5  msg:    .asciiz "Os oito primeiros numeros primos sao: "
6  space:  .ascii  " "
7
8  .text
9          la $t0, primos # Carrega o endereco inicial do array de primos
10         la $t1, size # Carrega o endereco do size
11
12         lw $t1, 0($t1) # Carrega size em t1
13
14         li $v0, 4 # Impressao da msg
15         la $a0, msg
16         syscall
17
18 loop:
19         beq $t1, $zero, exit # Se percorreu todo o array, encerra
20
21         li $v0, 1 # Impressao de um inteiro
22         lw $a0, 0($t0) # Inteiro a ser exibido
23         syscall
24
25         li $v0, 4 # Impressao do space
26         la $a0, space
27         syscall
28
29         addi $t0, $t0, 4 # Incrementa indice array
30         addi $t1, $t1, -1 # Decrementa contador
31         j loop
32
33 exit:
34         li $v0, 10 # Termino do programa
35         syscall
```

Ao executar o código no montador *MARS*, os dados e as instruções gerados, exibidos a seguir em hexadecimal, são ilustrados nas Figuras 1 e 2. Os dados gerados serão armazenados a partir do endereço $0x00002000$ na memória simulada, enquanto as instruções a partir do endereço $0x00000000$.

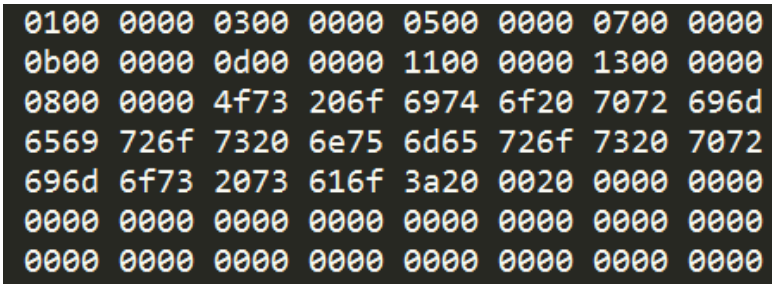

Figura 1: Conteúdo do arquivo gerado *data.bin*.

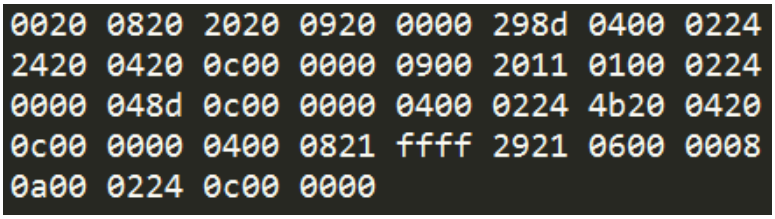

Figura 2: Conteúdo do arquivo gerado *text.bin*.

2 Descrição das Funções Implementadas

Além das funções elaboradas no trabalho 1, as seguintes funções foram implementadas para o funcionamento pleno do simulador *MIPS*.

2.1 Registradores

Os registradores *pc*, *ri* foram declarados como *unsigned*, pois o *pc* armazena um endereço, e o *ri*, uma instrução. Já os registradores *hi* e *lo* foram declarados como *signed*, pois estes podem possuir contéudos númericos dentro de si.

O banco de registradores *reg* é um vetor de 32 posições, representando os 32 registradores do *MIPS*, declarado com sinal, pois os registradores contém conteúdos númericos.

Os campos da instrução: *opcode*, *rs*, *rt*, *rd*, *shamt*, *funct* foram declarados todos como *unsigned*. Para intruções do tipo I, um campo *k16* também foi declarado como *signed* e para instruções do tipo J, outro campo, *k26*, foi declarado como *unsigned*.

2.2 *fetch(uint32_t endereco)*

A função *fetch()* le uma instrução da memória e a coloca em *ri*, atualizando o *pc* para apontar para a próxima instrução.

2.3 *decode()*

A função *decode()* decodifica a instrução armazenada em *ri* em campos, sendo estes campos *opcode*, *rs*, *rt*, *rd*, *shamt*, *funct*, *k16* e *k26*. Esta decodificação é realizada por meio de deslocamentos lógicos de bits, por exemplo, para extrair o campo *opcode* de uma instrução, sabendo que este campo possui 6 bits e a instrução inteira tem 32 bits, *ri* é deslocado 26 bits para a direita, resultando no campo *opcode*.

2.4 *execute()*

A função *execute()* executa a instrução lida pela *fetch()* e decodificada pela *decode()*. De acordo com os campos da instrução *opcode* e *funct*, a instrução é identificada e executada, por exemplo, se o *opcode* obtido de *ri* for $0x23$, em hexadecimal, a *green sheet* (ilustrada em parte na figura 3) do *MIPS* diz que a função a ser executada é um *load word*.

MIPS Reference Data ①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

Figura 3: Parte da *green sheet* utilizada de referência na função *execute()*.

2.5 *step()*

A função *step()* executa uma instrução do *MIPS*: *fetch(pc)*, *decode()*, *execute()*.

2.6 *run()*

A função *run()* executa o programa até encontrar um *syscall* de encerramento, ou até o *pc* atingir o fim da memória.

2.7 *dump_mem(uint32_t start, uint32_t end, char format)*

Imprime o conteúdo da memória a partir do endereço *start* até o endereço *end*, ambos em indice do vetor de memória. O formato pode ser em hexadecimal (padrão), ou decimal 'd'.

2.8 *dump_reg(char format)*

Imprime o conteúdo do banco de registradores e dos registradores *pc*, *hi*, *lo* do simulador *MIPS*. O formato pode ser em hexadecimal (padrão), ou decimal 'd'.

3 Testes e Resultados

Os testes que englobam todas as instruções foram desenvolvidos nos arquivos *.asm* e *.bin* disponíveis no diretório *Testes*. Também foram testados os exercícios do Laboratório 1, cujos arquivos *.asm* e *.bin* estão disponíveis no diretório *Laboratório*. E finalmente os arquivos *exemplo.asm* e *fibonacci.asm* representam os testes requeridos na especificação do trabalho.