Arquitetura e Organização de Computadores I

Trabalho prático 2

Heitor Lourenço Werneck
heitorwerneck@hotmail.com

26 de Junho de 2019

1 Introdução

O objetivo desse trabalho é apresentar um processador MIPS construído com as instruções R-format, beq, lw, sw, addi e j.

Uma CPU tem como principais componentes: registradores, ULA(Unidade lógica aritmética) e unidade de controle.

Para ver como a CPU se comporta com as instruções e permitir uma flexibilidade maior nos algoritmos também irá haver uma memória que a CPU poderá interagir.

As instruções implementadas junto com seus opcodes respectivos são mostradas na tabela 1.

Instrução	opcode
R-format	000000
lw	100011
sw	101011
beq	000100
addi	001000
j	000010

Tabela 1: Instruções implementadas.

2 ALU

Uma ALU foi criada com as instruções da tabela 2.

Controle da ALU	Operação
0000	AND
0001	OR
0010	ADD
0110	SUB
0111	Set on less than
1100	NOR

Tabela 2: Operações implementadas na ALU.

O esquemático da ALU é mostrado na figura 1.

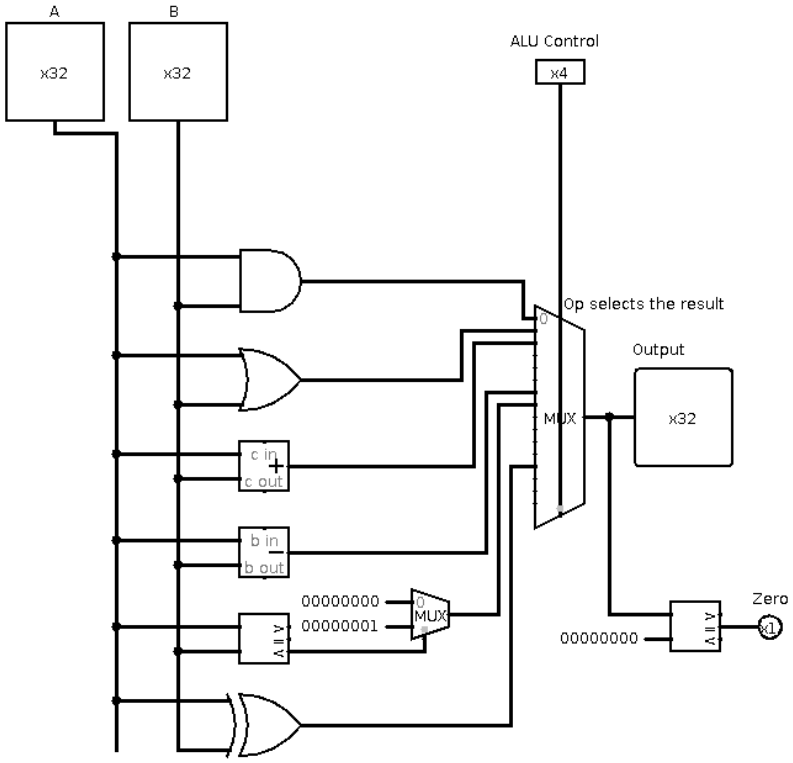

Figura 1: ALU.

3 Decodificador de tipo

Para saber qual tipo de instrução um opcode representa foi criado um decodificador de tipo para as instruções da tabela 1. A figura 2 mostra o esquemático.

4 Decodificador de controle

Com o decodificador de tipo de instrução basta ativar os sinais de controle de acordo com a instrução de entrada. Essa decodificação para sinais de controle é feita de acordo com a tabela 3.(Sendo X um *don't care*)

A figura 3 mostra o esquemático.

	R-format	lw	sw	beq	addi	j
RegDest	1	0	X	X	0	0
AluSrc	0	1	1	0	1	0
MemtoReg	0	1	X	X	0	0
RegWrite	1	1	0	0	1	0
MemRead	0	1	0	0	0	0
MemWrite	0	0	1	0	0	0
Branch	0	0	0	1	0	0
ALUOp1	1	0	0	0	0	0
ALUOp0	0	0	0	1	0	0
Jump	0	0	0	0	0	1

Tabela 3: Sinais de controle de acordo com instrução.

O addi tem AluSrc=1 para que o valor de [15..0](offset) da instrução seja somado ao valor do registrador, o RegWrite=1 é para que o valor seja escrito no registrador destino e o RegDst=0

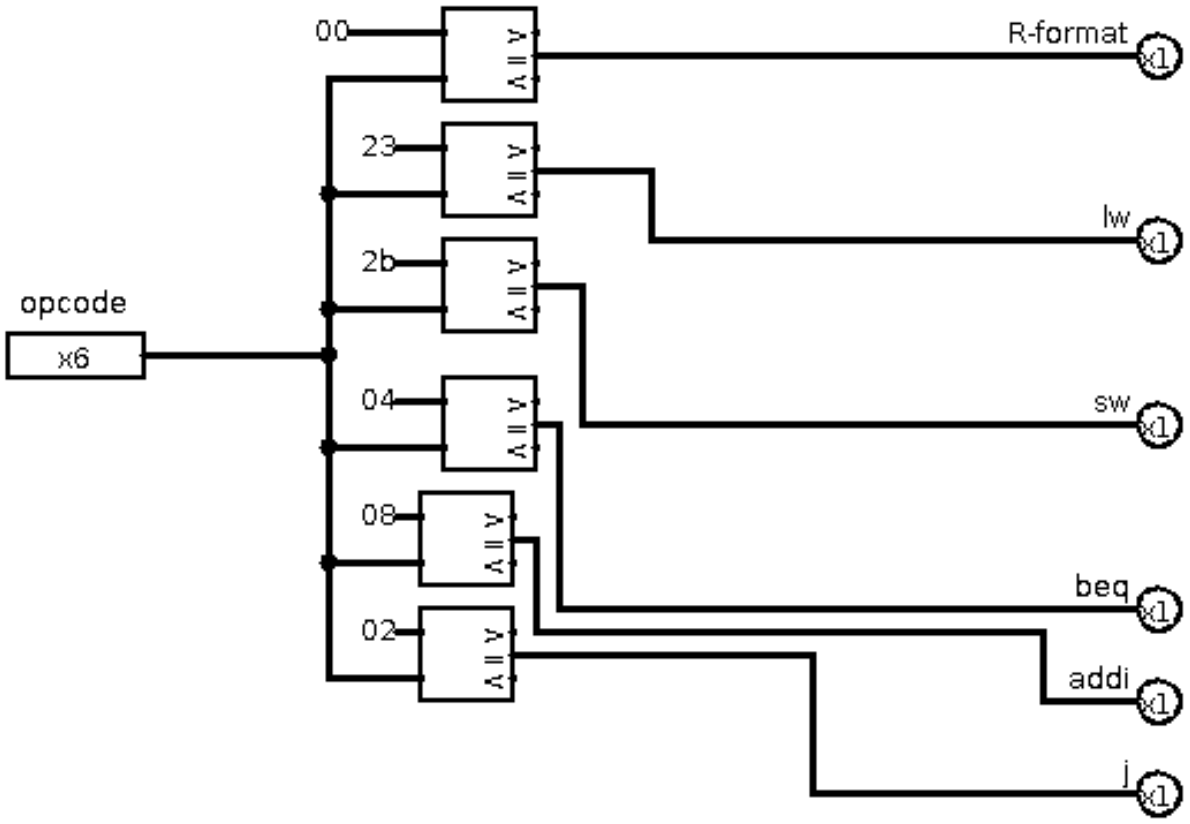

Figura 2: Decodificador de tipo.

para que o registrador que será escrito seja o [20..16](rt).(figura 4)

Para a instrução j foi criado um sinal de controle exclusivo chamado Jump que desencadeia as operações necessárias para que a instrução ocorra.

5 Unidade de controle

Com o decodificador de controle e decodificador de tipo basta juntá-los, o resultado disso é a figura 5.

6 Controle da ALU

Cada tipo de instrução usa a ALU de alguma maneira diferente logo foi criado um controle de operações da ALU com base na tabela 4. Então de acordo com o campo funct da instrução e o sinal ALUOp o tipo de operação da ALU é escolhido. O esquemático do controle da ALU é dado na figura 6.

7 Caminho de dados

O ultimo passo é a construção do caminho de dados. Na figura 7 há o caminho de dados. É necessário interpretar as consequências de cada sinal de controle.

O RegDst controla qual registrador será escrito, será o endereço do registrador no campo rd se RegDst=1 ou o campo rt se RegDst=0.

Com RegWrite=1 o registrador que será escrito receberá os dados da entrada de escrita.

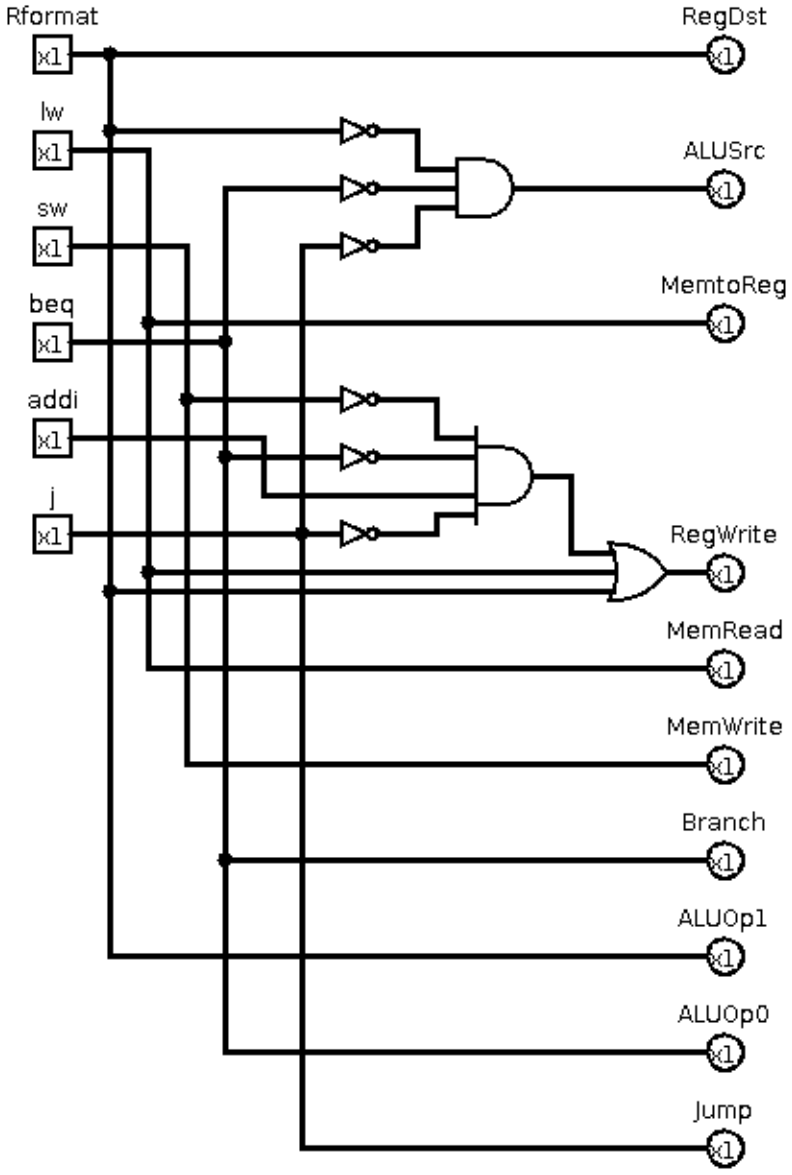

Figura 3: Decodificador de controle.

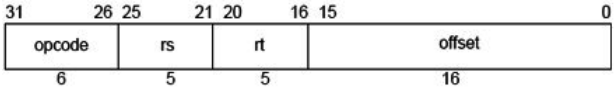

Figura 4: Instrução tipo I.

ALUOp	Campo Funct	Operação	
00	XXXXXX	0010	ADD
X1	XXXXXX	0110	SUB
1X	XX0000	0010	ADD
1X	XX0010	0110	SUB
1X	XX0100	0000	AND
1X	XX0101	0001	OR
1X	XX1010	0111	Set on less than

Tabela 4: Controle da ALU.

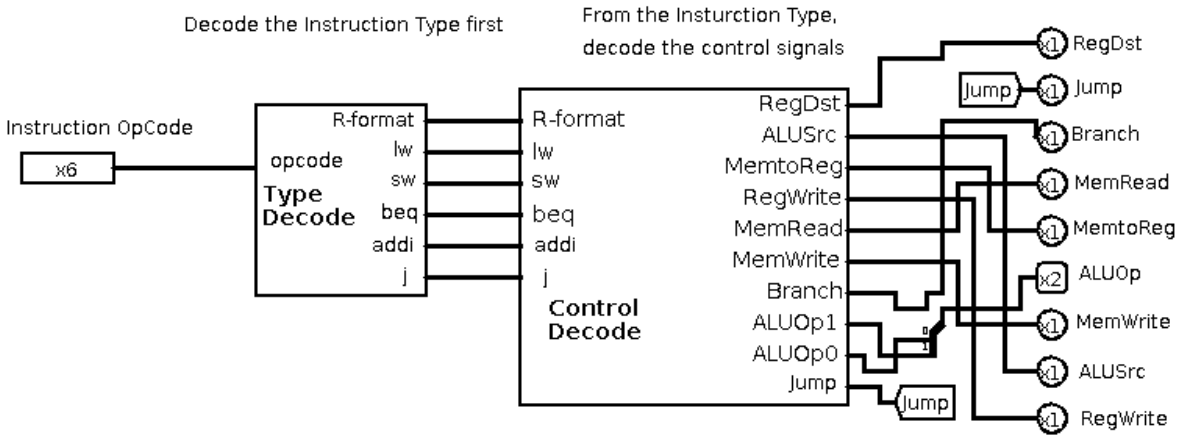

Figura 5: Unidade de controle.

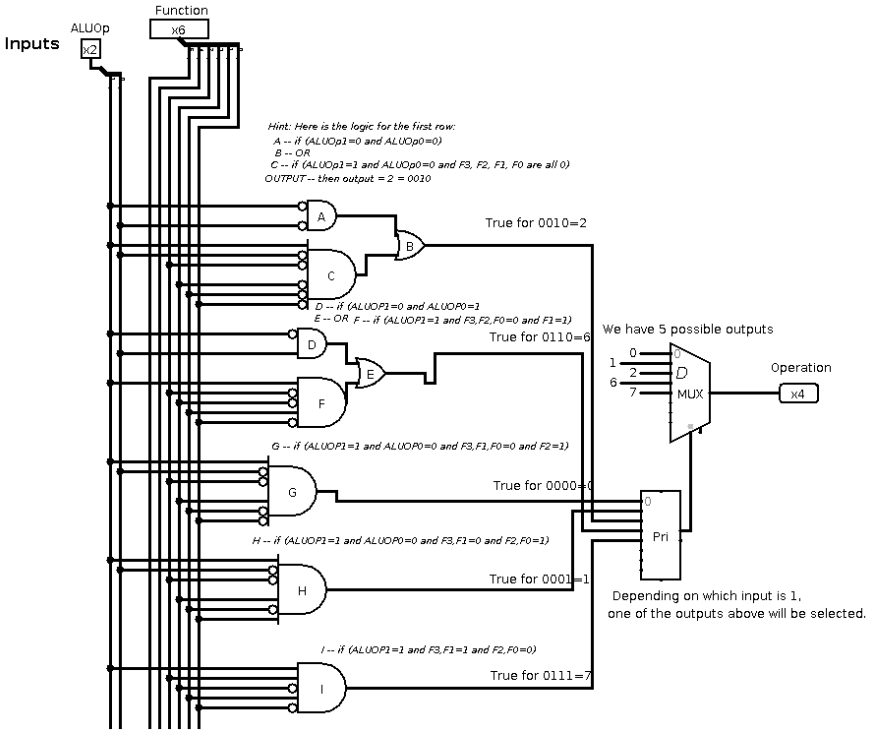

Figura 6: Controle da ALU.

Com ALUSrc=0 o segundo operando da ALU será o segundo registrador lido. Já com ALUSrc=1 o segundo operando da ALU será os 16 bits de baixo da instrução atual([15:0]).

Com Branch=0 ou com os valores dos registradores diferentes o valor do PC(Program counter) será mudado para PC+4. Com Branch=1 e valores nos registradores iguais o valor de PC será substituído pelo offset da instrução deslocado 2 bits para a esquerda somado com o valor de PC+4. Porém se o Jump=1 então o valor de PC será substituído pelo valor obtido da lógica do Jump.

A lógica do Jump é de pegar os 26 bits do campo de imediato da instrução[25:0] e esses 26 bits com deslocamento de 2 para a esquerda serem a parte de [27:0] do endereço final e os 4 bits mais significativos de PC+4 serão a parte [31:28] do endereço final resultante.

Se MemRead=1 então com o endereço informado na entrada de endereço da memória é obtido o valor correspondente a esse endereço. Se MemWrite=1 então o conteúdo do endereço informado é substituído pelo valor na entrada de escrita da memória.

Se MemtoReg=0 então o valor indo para a entrada de dados de escrita do registrador vem da ALU, se não então o valor vem da memória.

As consequências do valor do ALUOp é descrito na tabela 4.

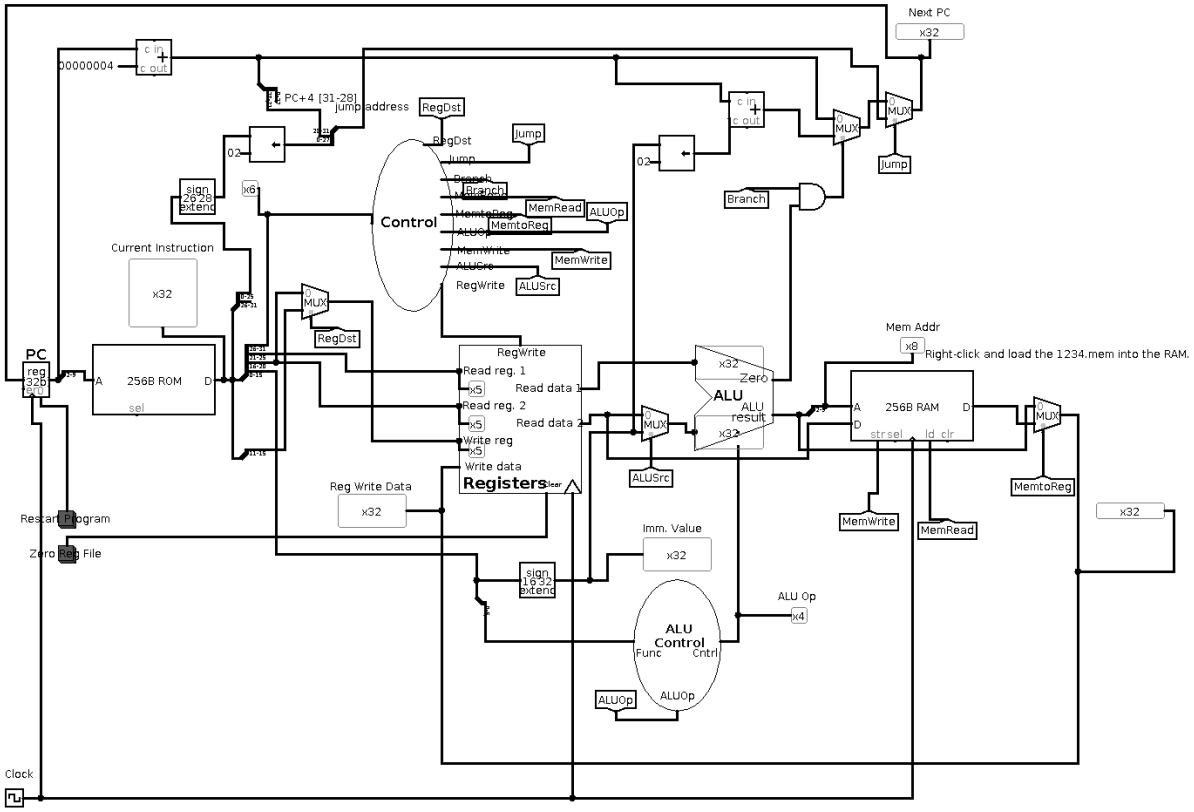

Figura 7: Caminho de dados.

8 Validação

Para fazer a validação do processador criado foi utilizado o Mars 4.5 para fazer a transformação do código em assembly MIPS para hexadecimal. O código a seguir deve retornar a soma $\sum_{i=1}^{a0} i$.

```
lw $t0,0($zero)
# cumulative sum f(a0)
add $t1,$zero,$zero
sum_loop:
beq $t0,$zero,sum_exit
add $t1,$t1,$t0
```

```
addi $t0,$t0,-1
j sum_loop
sum_exit:
sw $t1,4($zero)
```

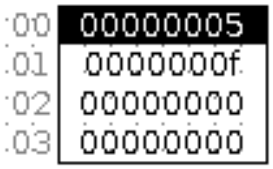

Figura 8: Exemplo de entrada e saída.(Entrada na primeira linha e saída na segunda)