Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Apucarana

Engenharia De Computação

ARCO3A - Arquitetura e Organização de Computadores

Simulador MIPS

Deivid Da Silva Galvão

Eduardo Yuji Yoshida Yamada

André D'Amato

Apucarana-PR, 2023

SUMÁRIO

1. INTRODUÇÃO

Este projeto consistiu na criação de um simulador MIPS para executar instruções em uma CPU-RISC de 16 bits. A simulação ocorreu em um modo de operação passo a passo, utilizando a linguagem de programação Python como base de implementação.

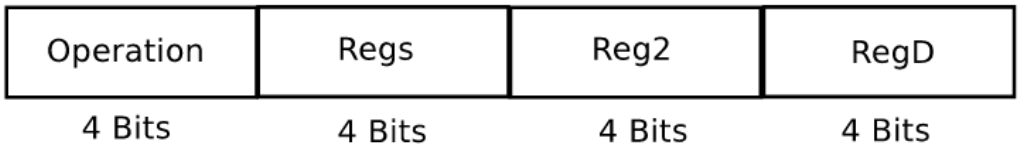
Especificações do Simulador:

1. O simulador deverá abrir arquivos binários;
2. Executar as instruções: lógicas, Aritméticas, desvios, transferência e acesso a memória;
3. Exibir o estado dos registradores, através de uma interface mostrando a execução das instruções, que ocorre passo a passo, onde as instruções serão executadas uma a uma mediante a interação com o usuário, enquanto informa as instruções executadas e o estado dos registradores durante, e após a execução;
4. Executar o programa de teste
5. Contar com conjunto de instruções:

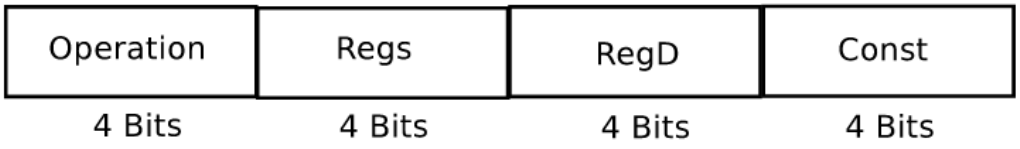
Categoria	Instrução	Opcode	Exemplo
Aritmética	Add	0010_2	Add $s1,$s2,$s3
	Sub	0011_2	sub $s1,$s2,$s3
	Addi	1000_2	addi $s1,100
	Shift	1001_2	Sft $s1,8
Lógica	And	0100_2	And $s1,$s2,$s3
	Or	0101_2	or $s1,$s2,$s3
	Not	1010_2	Not $s1
	Xor	0110_2	xor $s1,$s2,$s3
	Slt	0111_2	Slt $s1,$s2,$s3
Transferência	Lw	0000_2	lw $s1,$s2,$s3
	Sw	0001_2	sw $s1,$s2,$s3
	Lui	1011_2	Lui $s1,100
Desvio Condicional	Beq	1100_2	beq $s1,$s2,5
	Blt	1101_2	blt $s1,$s2,5
Desvio incondicional	J	1110_2	J $s1,100
	Jal	1111_2	Jal $s1,100

- Seguir os formatos de instruções:

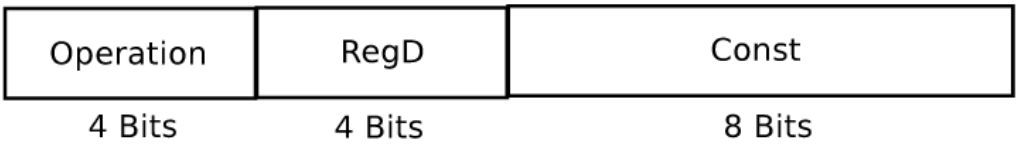
R Type



I Type



J Type



- Considerar os seguintes registradores:

Código	Símbolo	Função	Descrição
0000_2	$zero	Constante zero	Constante 0 de 16 bits
0001_2 0010_2 0011_2	$t0 $t1 $t2	Temporários	Registradores Auxiliares
0100_2 0101_2 0110_2	$a0 $a1 $a2	Argumento	Argumentos para operações aritméticas e procedimentos
0111_2 1000_2 1001_2 1010_2 1011_2	$s0 $s1 $s2 $s3 $s4	salvos	Armazena valores durante chamadas de procedimento
1100_2	$gp	Apontador global	Aponta para as variáveis globais na pilha
1101_2	$sp	Apontador pilha	Aponta para o topo da pilha
1110_2	$pc	Contador de programa	Aponta para a próxima instrução
1111_2	$ra	Endereço de Retorno	Armazena o endereço de retorno de uma rotina

2. LÓGICA GERAL DE FUNCIONAMENTO

Dentro da função "carregarInstrucoes", o conteúdo do arquivo de entrada será lido. Cada linha do arquivo representa uma instrução, que será dividida em quatro partes. Essas partes são usadas para identificar o código de operação (opcode) e os operadores da instrução. Cada linha, após ser segmentada, é armazenada em uma lista chamada "instrucoes". Em seguida, a função retorna para a função principal. Usando a função "len()", calculamos o número de instruções multiplicado por quatro, que representa o número máximo de instruções que podem ser armazenadas no PC. A partir disso, a função principal chama a função “separaInstrucoes”. Essa função analisa o opcode para identificar o tipo de instrução que deve ser executada e a encaminha para a função correspondente no código. Dentro de cada função correspondente a um tipo de instrução, verificamos o valor do opcode e executamos a operação adequada com base nesse valor.

executaInstrucaoR: As funções de tipo R chamam a função “recebeRegistro”, que é responsável por identificar os registradores que serão utilizados na operação, realizando assim, a operação com os valores dos registradores de saída e auxiliar, e então, armazenando o resultado no registrador de destino, através da função “modificaRegistro”. A função add soma os valores dos registradores de saída e auxiliar, e o armazena no registrador de destino. A função sub funciona da mesma forma que o add, porém ao invés de somar os valores, ela subtrai. Já as operações lógicas or, xor e and fazem a comparação entres os valores dos registradores de saída e auxiliar, e armazenam o valor em booleano da comparação no registrador de destino.

executaInstrucaoI: Dentro das funções de tipo I, os valores dos registradores e do inteiro são recebidos pela função “recebeRegistro”, onde seu opcode é verificado e as devidas operações são realizadas. A função addi altera o valor dentro do registrador através da função “somaRegistro”, responsável por somar o valor do inteiro com o valor armazenado no registrador de saída, armazenando o valor final no registrador de destino. A função shift altera o valor do registrador de saída, pulando uma casa lógica para a direita e inserindo esse valor dentro do registrador de destino através da função “modificaRegistro”. Já as funções bne e beq comparam o valor dos registradores de saída e de destino, verificando se a comparação falhou ou se foi bem sucedida, em caso de sucesso, ela pula para a posição de pc, calculada através do valor do inteiro vezes 4.

executaIntrucaoJ: Assim como nas outras funções, as de tipo J recebem os valores do registrador e do inteiro através da função “recebeRegistro”. As funções jump e jal pulam para um ponto específico na pc, que é calculado através da soma do valor dentro do registrador de destino e do inteiro, tudo isso multiplicado por 4, sendo que, na função jal, ele salva esse valor dentro do vetor ra. Dentro da função "executaIntrucaoJ" também está a função nor, que obtém o valor dentro do registrador de destino e inverte seu valor em binário.

3. IMPLEMENTAÇÃO

Todas as funções requisitadas pelo professor foram implementadas dentro do código e testadas adequadamente.

Exemplos de funcionamento:
1. Código de Fatorial:
entrada em binário:

```
1000000000010000    addi $t0, $t0, 0
1000000000100001    addi $t1, $t1, 1
1000000001110101    addi $s0, $s0, 5
1000000010010001    addi $s2, $s2, 1
1100011100011111    beq $s0, $zero, sair
0010011100011000    add $s0, $t0, $s1
0010000100011010    addi $s3, $s3, 0
1100100000011100    beq $s1, $zero, sair2
0010101010011010    add $s3, $s3, $s2
0011100000101000    sub $s1, $t1, $s1
1110000000001000    j loop2
0010000110101001    add $s2, $t0, $s3
0011011100100111    sub $t0, $t0, $s0
1110000000000101    j loop
0010000100011010    add $s3, $t0, $t0
```

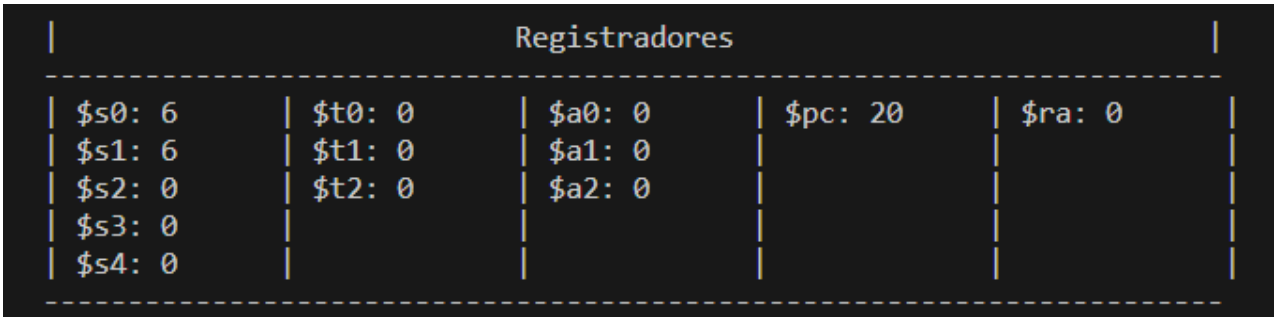
Saída:

Registradores				
$s0: 0	$t0: 0	$a0: 0	$pc: 64	$ra: 0
$s1: 0	$t1: 1	$a1: 0		
$s2: 120	$t2: 0	$a2: 0		
$s3: 0				
$s4: 0				

2. Código de Soma e Subtração:
entrada em binário:

```
1000000001111010    addi $s0, $zero, 10
1000000010000110    addi $s1, $zero, 6
0010100010010111    add $s1, $s2, $s3
0011011110001001    sub $s0, $s1, $s4
```

Saída:



3. Código com loop:
entrada em binário:

1000000001110101	addi $zero, $s0, 5
1000000010000001	addi $zero, $s1, 1
1100011110000101	beq $s0, $s1, inverte
1110111000000010	jump loop
1010011100000000	not $s0

Saída: