2014.2

UFRRJ IM

Gabriel Segobia
Paulo Xavier
Reinaldo Moraes

[LABORATÓRIO DE ARQUITETURA DE COMPUTADORES I]

Atividade Acadêmica de Arquitetura de Computadores, que consiste em um simulador MPIS para web, programado em Java-Web. O simulador recebe um código em Assembly, o monta e executa o mesmo, exibindo o código de maquina resultante.

Sumário

1 Descrição 2
1.1 Abrir 2
1.2 Salvar 2
1.3 Montar 2
1.4 Executar tudo 2
1.5 Executar passo a passo 2
2 Instruções 2
3 Abas Editar e Executar 6
3.1 Editar 6
3.2 Executar 6
4 Memória 6
5 Registradores 7
6 Caixa de status 7
7 Pesquisa 8

1 Descrição

O projeto consiste na elaboração de um software desenvolvido na linguagem Java-Web que recebe um código em Assembly, o transforma em código binário e simula a execução dessas instruções convertidas em linguagem de máquina, respeitando o padrão da arquitetura MIPS.

A entrada do código que será convertido é feita pela tela principal da aplicação e os comandos para abrir, salvar, montar, executar tudo e executar passo a passo estão disponíveis acima dessa caixa de texto e a exibição do código de máquina aparecerá ao lado.

1.1 Abrir

A função “abrir” abre uma caixa para seleção de um arquivo de texto para ser carregado para a tela de edição, onde poderá ser montado e executado. Essa função pode ser executada pelas teclas de atalho (Ctrl+F1).

1.2 Salvar

A função “salvar” armazena o código em assembly em um arquivo no formato “.txt” que posteriormente pode ser acessado novamente. Essa função pode ser executada pelas teclas de atalho (Ctrl+F2).

1.3 Montar

A função “montar” converte o código em Assembly na caixa de texto para linguagem de máquina, as transforma em instruções prontas para serem executadas e as exibe ao lado. Essa função pode ser executada pelas teclas de atalho (Ctrl+F3).

1.4 Executar tudo

Esta função executa todo o código que foi montado anteriormente, ou seja, usa as instruções em código de máquina que já foram montadas para executar o código. Essa função pode ser executada pelas teclas de atalho (Ctrl+F6).

1.5 Executar passo a passo

Esta função, assim como a função acima, executa o código q foi montado anteriormente, mas desta vez, com uma instrução de cada vez. Essa função pode ser executada pelas teclas de atalho (Ctrl+F7).

2 Instruções

O programa tem como objetivo executar as instruções da arquitetura MIPS, que estão listadas abaixo com suas respectivas funções, categorias, nomes, sintaxes, significado e formato.

Categoria	Nome	Sintaxe da Instrução	Significado	Tipo	Função
Aritmética	Add	add $s1,$s2,$s3	$s1 = $s2 + $3	R	Adiciona dois registros, estende sinal de largura de registro.
	Add unsigned	addu $s1,$s2,$s3	$s1 = $s2 + $s3	R	Como acima, sem extensão de sinal de largura de registro.
	Subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	R	Subtrai dois registradores.
	Subtract unsigned	subu $s1,$s2,$s3	$s1 = $s2 – $s3	R	Como acima, sem extensão de sinal de largura de registro.
	Add immediate	addi $s1,$s2,CONST	$s1 = $s2 + CONST	I	Soma um registrador a uma constante.
	Add immediate unsigned	addiu $s1,$2,CONST	$s1 = $s2 + CONST	I	Como acima, sem extensão de sinal de largura de registro.
Transferência de Dados	Load word	lw $s1,CONST($s2)	$s1 = Memory[$s2 + CONST]	I	Carrega o termo armazenado a partir de: MEM[$s2+CONST] e os seguintes 3 bytes.
	Load halfword	lh $s1,CONST($s2)	$s1 = Memory[$s2 + CONST] (signed)	I	Carrega meio termo armazenado a partir de: MEM[$s2+CONST] e o byte seguinte, estende sinal de largura de registro.
	Load byte	lb $s1,CONST($s2)	$1 = Memory[$2 + CONST] (signed)	I	Carrega o byte armazenado: MEM[$s2+CONST].

	Store word	sw $s1,CONST($s2)	Memory[$s2 + CONST] = $s1	I	Armazena um termo em: MEM [$s2+CONST] e os seguintes 3 bytes.
	Store half	sh $s1,CONST($s2)	Memory[$s2 + CONST] = $s1	I	Armazena a primeira metade do um registo (uma meio termo) em: MEM[$s2+CONST] e o byte seguinte.
	Store byte	sb $s1,CONST($s2)	Memory[$s2 + CONST] = $s1	I	Armazena o primeiro quarto de um registro (um byte) em: MEM [$s2+CONST].
	Load upper immediate	lui $s1,CONST	$s1 = CONST << 16	I	Carrega um operador imediato de 16 bits para os 16 bits superiores do registro especificado.
Lógico	And	and $s1,$s2,$s3	$s1 = $s2 & $s3	R	Executa a operação lógica AND bit a bit entre dois registradores.
	And immediate	andi $s1,$s2,CONST	$s1 = $s2 & CONST	I	Executa a operação lógica AND bit a bit entre um registrador e uma constante.
	Or	or $s1,$s2,$s3	$s1 = $s2 | $s3	R	Executa a operação lógica OR bit a bit entre dois registradores.
	Or immediate	ori $s1,$s2,CONST	$s1 = $s2 | CONST	I	Executa a operação lógica OR bit a bit entre um registrador e uma constante.
	Exclusive or	xor $s1,$s2,$s3	$s1 = $s2 ^ $s3	R	Executa a operação lógica XOR bit a bit entre dois registradores.
	Exclusive or	xori $s1,$s2,CONST	$s1 = $s2 ^ CONST	I	Executa a operação lógica XOR bit a

	unsigned				bit entre um registrador e uma constante.
	Nor	nor $s1,$s2,$s3	$s1 = ~($s2 | $s3)	R	Executa a operação lógica NOR bit a bit entre dois registradores.
	Set on less than	slt $s1,$s2,$s3	$s1 = ($s2 < $s3)	R	Testa se o primeiro registrador é menor que o segundo.
	Set on less than immediate	slti $s1,$s2,CONST	$s1 = ($s2 < CONST)	I	Testa se o registrador é menor que uma constante.
Deslocamento bit a bit	Shift left logical	sll $s1,$s2,CONST	$s1 = $s2 << CONST	R	Desvia para esquerda o número de casas da constante.
	Shift right logical	srl $s1,$s2,CONST	$s1 = $s2 >> CONST	R	Desvia para direita o número de casas da constante.
Desvio condicional	Branch on equal	beq $s1,$s2,CONST	if ($s1 == $s2) go to PC+4+CONST	I	Desvia o número de instruções definido pela constante se os dois registradores forem iguais.
	Branch on not equal	bne $s1,$s2,CONST	if ($s1 != $s2) go to PC+4+CONST	I	Desvia o número de instruções definido pela constante se os dois registradores forem diferentes.
salto incondicional	Jump	j CONST	goto address CONST	J	Desvia o numero de linhas da constante ou vai para label desejada. Pula o numero de linhas da constante ou vai para label desejada.
	Jump register	jr $s1	goto address $s1	R	Desvia para o endereço do registrador.
	Jump and link	jal CONST	$s31 = PC + 4; goto CONST	J	Desvia o numero de instruções da constante ou vai para label desejada e altera o valor de $ra para PC + 4.

3 Abas Editar e Executar

A janela principal da aplicação é dividida em duas abas, editar e executar, onde o usuário entra com o código e vê as instruções que serão executadas.

3.1 Editar

É uma caixa de texto que recebe o código em Assembly que poderá ser montado, salvo e executado pelo usuário.

```
1  addi $t0, $t0, 10
2
3  label:
4
5  addi $t1, $t1, 1
6
7  bne $t0, $t1, label
```

3.2 Executar

Aba onde aparecem as instruções reconhecidas após a montagem e o código binário gerado por elas.

Instrução	Codigo Binario
addi $t0, $t0, 10	00100001000010000000000000001010
addi $t1, $t1, 1	00100001001010010000000000000001
bne $t0, $t1, label	00010101000010011111111111111111

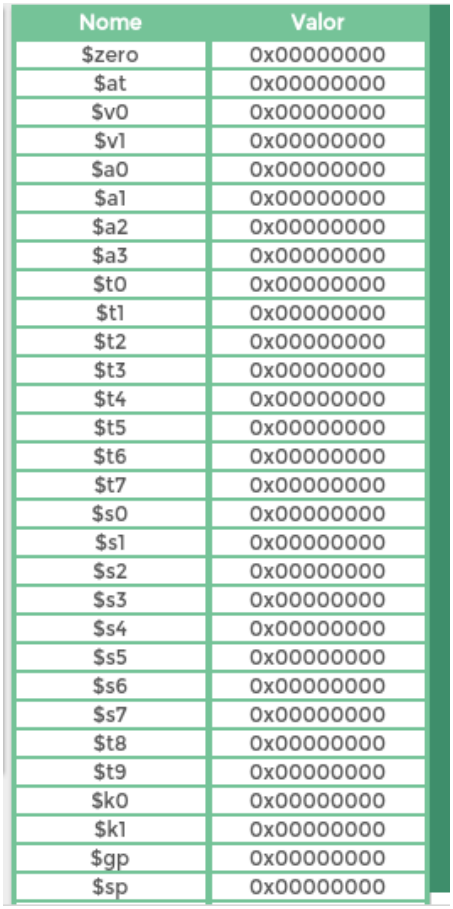
4 Memória

Na parte inferior da página, existe um campo que exibe o valor de cada posição da memória para visualização do usuário.

Endereço Base	+0	+1	+2	+3
0	00000000	00000000	00000000	00000000
4	00000000	00000000	00000000	00000000
8	00000000	00000000	00000000	00000000
12	00000000	00000000	00000000	00000000
16	00000000	00000000	00000000	00000000
20	00000000	00000000	00000000	00000000
24	00000000	00000000	00000000	00000000
28	00000000	00000000	00000000	00000000
32	00000000	00000000	00000000	00000000
36	00000000	00000000	00000000	00000000
40	00000000	00000000	00000000	00000000
44	00000000	00000000	00000000	00000000
48	00000000	00000000	00000000	00000000

5 Registradores

Exibe os todos os registradores com seu nome e respectivo valor que é atualizado em tempo de execução. Por exemplo, usando o modo “executar passo-a-passo”, onde os registradores são atualizados a cada instrução executada.



Nome	Valor
$zero	0x00000000
$at	0x00000000
$v0	0x00000000
$v1	0x00000000
$a0	0x00000000
$a1	0x00000000
$a2	0x00000000
$a3	0x00000000
$t0	0x00000000
$t1	0x00000000
$t2	0x00000000
$t3	0x00000000
$t4	0x00000000
$t5	0x00000000
$t6	0x00000000
$t7	0x00000000
$s0	0x00000000
$s1	0x00000000
$s2	0x00000000
$s3	0x00000000
$s4	0x00000000
$s5	0x00000000
$s6	0x00000000
$s7	0x00000000
$t8	0x00000000
$t9	0x00000000
$k0	0x00000000
$k1	0x00000000
$gp	0x00000000
$sp	0x00000000

6 Caixa de status

Abaixo do campo de editar e executar, está a caixa de status que exibe mensagens para orientação do usuário, como mensagens de erro e de confirmação, caso o código seja montado com sucesso.

7 Pesquisa

Localizada no canto superior direito, a caixa de pesquisa funciona como ajuda ao usuário. Pesquisando mnemônicos, ela abra uma janela com o nome, sintaxe e descrição do mesmo.

Neste caso, simulamos a busca pelo mnemônico “add”.