Material de Consulta

Registradores

Notação	Número	Descrição
`$zero`	0	A constante zero
`$at`	1	Reservado para o *assembler*
`$v0-$v1`	2-3	Valores para resultados e avaliação de expressões
`$a0-$a3`	4-7	Argumento
`$t0-$t7`	8-15	Temporários (não preservados entre chamadas)
`$s0-$s7`	16-23	Salvos (preservados entre chamadas)
`$t8-$t9`	24-25	Mais temporários
`$k0-$k1`	26-27	Reservado para o *kernel* do SO
`$gp`	28	ponteiro global
`$sp`	29	ponteiro para pilha
`$fp`	30	ponteiro para frame
`$ra`	31	endereço de retorno

Convenção de uso dos registradores MIPS.

Comando para executar código assembly

```
spim -f imprime_char.spim
```

Comandos li $v0

Comando	Significado
li $v0, 1	imprimir inteiro
li $v0, 2	imprimir float
li $v0, 3	imprimir double
li $v0, 4	imprimir String ou char
li $v0, 5	ler inteiro
li $v0, 6	ler float
li $v0, 7	ler double
li $v0, 8	ler String ou char
li $v0, 10	encerrar programa principal

Código Assembly comum para hello world:

```
.data
    #área para dados na memória principal
ola: .asciiz "Ola Mundo\n" #mensagem a ser exibida para o usuário

.text
    #área para instruções do programa

#Função a ser executada pelo compilador
main:

#Etapa de imprimir o resultado da String
li      $v0,     4          #instrução para impressão de String
la      $a0,     ola        #indicar o endereço em que está a mensagem
syscall                     #faça! imprima

# terminando a execução do programa
li      $v0,    10      #v0 = exit
syscall
```

System Calls

Table: System services.

Service	System Call Code	Arguments	Result
print_int	1	`$a0` = integer	
print_float	2	`$f12` = float	
print_double	3	`$f12` = double	
print_string	4	`$a0` = string	
read_int	5		integer (in `$v0`)
read_float	6		float (in `$f0`)
read_double	7		double (in `$f0`)
read_string	8	`$a0` = buffer, `$a1` = length	
sbrk	9	`$a0` = amount	address (in `$v0`)
exit	10		
print_character	11	`$a0` = character	
read_character	12		character (in `$v0`)
open	13	`$a0` = filename,	file descriptor (in `$v0`)
		`$a1` = flags, `$a2` = mode	
read	14	`$a0` = file descriptor,	bytes read (in `$v0`)
		`$a1` = buffer, `$a2` = count	
write	15	`$a0` = file descriptor,	bytes written (in `$v0`)
		`$a1` = buffer, `$a2` = count	
close	16	`$a0` = file descriptor	0 (in `$v0`)
exit2	17	`$a0` = value	

System calls

Serviço	Cod.	Argumentos	Resultado
imprimir inteiro	1	`$a0` = inteiro	n. a.
imprimir uma string	4	`$a0` = endereço da string	n. a.
ler um inteiro	5	n. a.	`$v0` = valor lido
ler uma string	8	`$a0` = endereço da string `$a1` = qtde. de caracteres + 1	n. a.
alocar memória	9	`$a0` = número de bytes	`$v0` = endereço do bloco
encerrar o programa	10	n. a.	n. a.
imprimir um caracter	11	`$a0` = inteiro (ASCII)	n. a.
ler um caracter	12	n. a.	`$v0` = caracter lido

Observação: as operações 2 e 3, 6 e 7 são operações com números de ponto flutuante, que veremos adiante.

Tipos de Dados

Escrevendo em MIPS

- Tipos de dados:
  - `.word` w_1, ..., w_n: dado de 32 bits
  - `.half` h_1, ..., h_n: dado de 16 bits
  - `.byte` b_1, ..., b_n: dado de 8 bits
  - `.ascii str`: cadeia de caracteres
  - `.asciiz str`: terminando com o caracter nulo

Comandos Condicionais

COMANDOS CONDICIONAIS

Comando	Significado	Pronúncia
beq $t1, $t2, label	Se $t1 for igual a $t2, execute a partir do rótulo label	branch if equal
bne $t1, $t2, label	Se $t1 for diferente de $t2, execute a partir do rótulo label	branch if not equal
blt $t1, $t2, label	Se $t1 for menor que $t2, execute a partir do rótulo label	branch if less than
bgt $t1, $t2, label	Se $t1 for maior que $t2, execute a partir do rótulo label	branch if greater than
ble $t1, $t2, label	Se $t1 for menor ou igual a $t2, execute a partir do rótulo label	branch if less or equal
bge $t1, $t2, label	Se $t1 for maior ou igual a $t2, execute a partir do rótulo label	branch if greater or equal

Set on Less Than

A Instrução SLT

SLT significa **Set on less Than**, ao pé da letra seria algo como comparar menor que, então essa instrução será muito utilizada em comparações entre registradores, para identificar quem tem o maior ou menor valor. A função desta instrução é comparar dois valores de dois registradores diferentes e atribuir o valor 1 a um terceiro registrador se o valor do primeiro registrador for menor que o valor do segundo registrador. Caso contrário, atribuir zero. A sintaxe é:

SLT registrador_temporário, registrador1, registrador2

O formato da instrução é:

OpCode	RS	RT	RD	SHAMT	FUNCT
Código da Operação	Registrador Temporário	Registrador a ser comparado 2	Registrador a ser comparado 1	não usado	código da operação aritmética
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Vamos supor a seguinte instrução MIPS:

SLT $t0, $s1, $s2

Isso é o mesmo que:

$st0 = $s1 < $s2

```c
if ( i < j )
      a = b + c;
else
      a = b - c;
```

Vamos fazer a compilação desse trecho de código em C para MIPS, seguindo o nosso roteiro padrão:

1. Linguagem de Montagem;
2. Linguagem de Máquina;
3. Representação e;
4. Código de Máquina.

Considere a = $s0, b = $s1, c = $s2, i = $s3, j = $s4.

a) Linguagem de Montagem

linha	código
1	slt $t0, $s3, $s4
2	bne $t0, $zero, ELSE
3	add $s0, $s1, $s2 *#a = b + c; (se $t0 <> 0)*
4	j Exit *#desvia para exit*
5	ELSE: sub $s0, $s3, $s4 *#a = b – c; (se $t0 = 0)*
6	Exit:

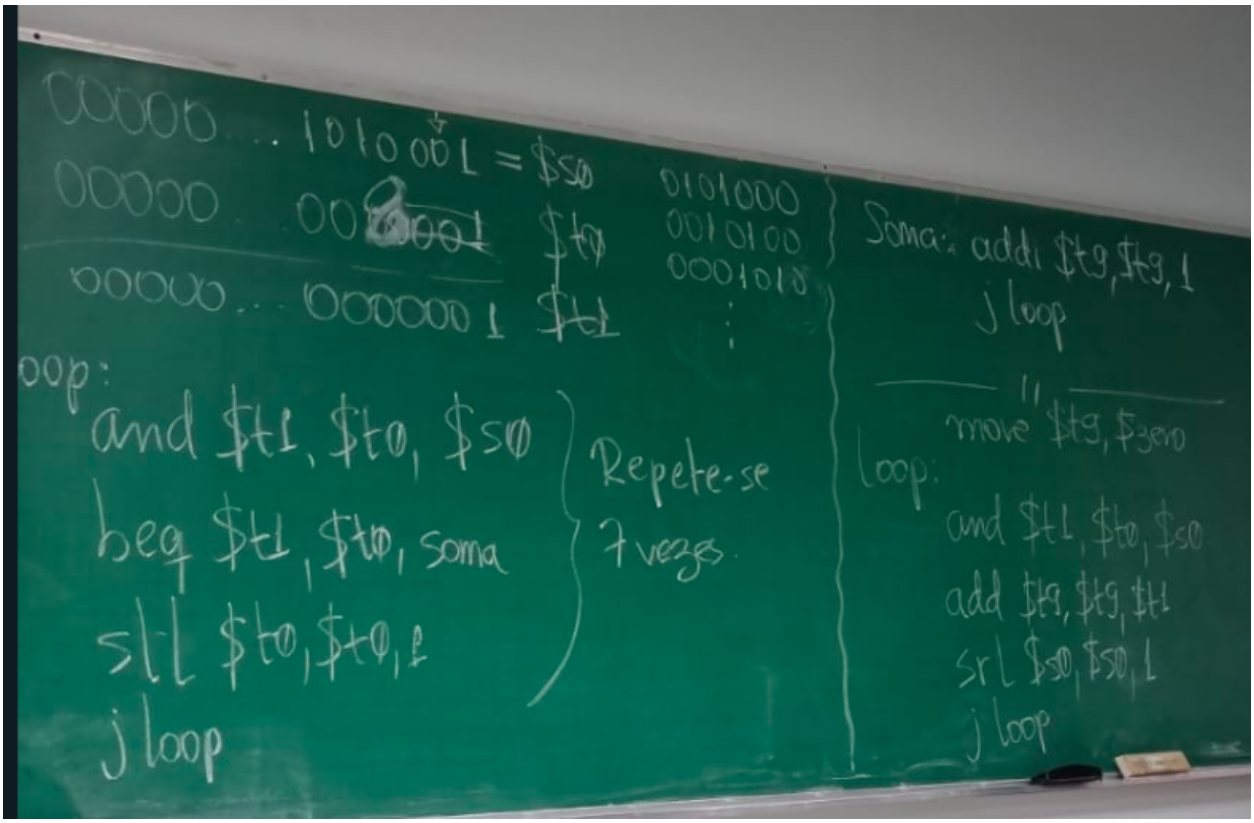
Pseudoinstruções

Cada pseudoinstrução pode ser traduzida em uma instrução, veja no exemplo abaixo a tradução do move e do li:

```
add $a0, $s0, $zero ;move

addi $v0, $zero, 9 ;li

addu ;add que considera numero unsigned (sem sinal)
```

Unsigned e Signed na comparação de SLT e SLTU

```
$s0 = 1011 = -5 ou 11 (se unsigned)
$s1 = 0101 = 5

slt $t0, $s0, $s1 -> $t0 = 1 pois -5 < 5
sltu $t0, $s0, $s1 -> $t0 = 0 pois 11 !< 5
```



```
1   bitparidade:
2       #$t0 => mascara
3       #$t1 => contador de interacoes
4       #$t9 => contador de bits 1
5
6       #inicializando v0 com 0
7       move    $v0, $zero
8
9       #v1 recebendo conteudo de a0
10      move    $v1, $a0
11
12      # incializando t0 = 1
13      li      $t0, 1
14
15      # inicializando t1 = 7
16      li      $t1, 7
17
18      # inicializando t9 = 0
19      move    $t9, $zero
20
21  ## $t1 = 0 encerro
22  loopOne: beq        $t1, $zero, exitOne
23
24      and     $t2, $t0, $a0
25
26      # t4 recebendo o conteudo de t0
27      move    $t4, $t0
28
29      #descola o bit da mascara
30      sll     $t0, $t0, 1         # $t0 = $t0 << 1
```

```
32      #contador de interacoes--
33      addi    $t1, $t1, -1            # $t1 = $t1 + -1
34
35      # Se o resultado do and for diferente da mascara, não temos bit 1
36      bne     $t4, $t2, loopOne
37
38      addi    $t9, $t9, 1         # $t9 = $t9 + 1
39
40      j loopOne
41
42  exitOne:
43
44      andi    $t8, $t9, 1         # $t8 = $t9 & 1
45      beq     $t8, $zero, returnOne
46
47      li      $v0, 1      # $v0 = 1
48
49      #128 = 10000000
50      ori     $v1, $a0, 128           # $v1 = $a0 | 128
51
52  returnOne:
53      jr      $ra                 # jump para $ra
54          You, anteontem • Exercicios A e B da segunda lista
```