

# Instruções: A Linguagem de Máquina

## Arquitetura RISC

## Instruções: A Linguagem de Máquina

- ▶ MIPS – (<http://www.mips.com/>) – Arquitetura RISC (Reduction Instruction Set)  
(**M**icroprocessor without **i**nterlocked **p**ipeline **s**tages)

- ▶ **Operações de hardware do computador**

Notação em assembly do MIPS

add a, b, c

Ex. soma das variáveis b, c, d, e e armazenar o resultado na variável a

add a, b, c    # a soma  $b + c$  é colocada em a

add a, a, d    # a soma  $b + c + d$  é armazenada em a

add a, a, e    # a soma  $b + c + d + e$  agora está armazenada em a

*Princípio de Projeto I: simplicidade favorece a regularidade.*

## Instruções: A Linguagem de Máquina

### ► Exemplo de compilação de instruções de atribuição C no MIPS

`a = b + c;`

`d = a - e;`

Assembly do MIPS

`add a,b,c`

`sub d,a,e`

## Instruções: A Linguagem de Máquina

### ► Compilando uma atribuição C complexa no MIPS

$f = (g + h) - (i + j)$

`add t0,g,h` # variável temporária t0 contém  $g + h$

`add t1,i,j` # variável temporária t1 contém  $i + j$

`sub f,t0,t1` # f recebe  $t0 - t1$ , que é  $(g + h) - (i + j)$

## Instruções: A Linguagem de Máquina

- ▶ **Operandos do hardware do Computador**

Registrador na arquitetura MIPS: 32 registradores de 32 bits

*Princípio de Projeto II : menor significa mais rápido.*

- ▶ **Compilando uma atribuição em C usando registradores**

$$f = (g + h) - (i + j)$$

## Instruções: A Linguagem de Máquina

### ► Compilando uma atribuição em C usando registradores

$$f = (g + h) - (i + j)$$

As variáveis  $f, g, h, i$  e  $j$  serão associadas aos registradores  $\$s0, \$s1, \$s2, \$s3$ , e  $\$s4$ , respectivamente.

```
add $t0,$s1,$s2    # registrador $t0 contém g + h
add $t1,$s3,$s4    # registrador $t1 contém i + j
sub $s0,$t0,$t1     # f recebe $t0 - $t1, que é (g + h) - (i + j)
```

## Instruções: A Linguagem de Máquina

### ► Compilando uma atribuição quando o operando está na memória

$g = h + A[8];$

Supondo que  $A$  é uma sequência de 100 words e que o compilador tenha associado as variáveis  $g$  e  $h$  aos registradores  $\$s1$  e  $\$s2$  e que o endereço inicial da sequência, ou endereço base, esteja em  $\$s3$ .

```
lw $t0, 8($s3)    # registrador temporário $t0 recebe A[8]
add $s1, $s2, $t0  # g = h + A[8]
```

A constante na instrução de transferência de dados é chamada de *offset*, e o registrador acrescentado para formar o endereço é chamado de *registrador base*.

## Instruções: A Linguagem de Máquina

### ► Interface hardware/software

A maioria das arquiteturas endereça bytes.

O endereço de uma word combina os endereços dos 4 bytes dentro da word.

No MIPS words precisam começar em endereços que seja múltiplos de 4 (restrição de alinhamento).

O endereçamento em bytes também afeta o índice do array, o offset somado ao registrador base `$s3` precisa ser 4x8.



## Instruções: A Linguagem de Máquina

### ➤ Organização da Memória

A maioria dos dados usam palavras ou "words"

Para MIPS, uma word é constituída de 32 bits ou 4 bytes.

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data

Os registradores carregam dados de  
32 bits

$2^{32}$  bytes têm endereços de 0 a  $2^{32}-1$

$2^{30}$  words têm endereços de bytes 0, 4, 8, ...  $2^{32}-4$

Words são alinhados

## Instruções: A Linguagem de Máquina

### ► Compilando com load e store

$A[12] = h + A[8]; \quad (h \text{ \$s2} \quad A \text{ \$s3})$

```
lw  $t0, 32($s3)
```

```
add $t0, $s2, $t0
```

```
sw  $t0, 48($s3)
```

### ► Soma imediata

```
addi $s3, $s3, 4    # $s3 = $s3 + 4
```

*Princípio de Projeto III : agilize os casos mais comuns.*

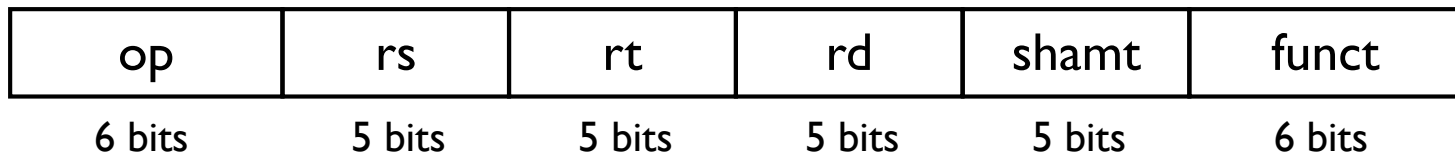
## Instruções: A Linguagem de Máquina

### ▶ MIPS - Registradores

- ▶ No assembly MIPS os registradores \$s0 a \$s7 são mapeados nos registradores 16 a 23
- ▶ Os registradores \$t0 a \$t7 são mapeados nos registradores de 8 a 15

## Instruções: A Linguagem de Máquina

### ► MIPS - Formato R



- Nome dos campos nas instruções
  - op: operação básica da instrução (opcode)
  - rs: registrador do primeiro operando de origem
  - rt: registrador do segundo operando de origem
  - rd: registrador do operando de destino
  - shamt: shift amount (00000 por enquanto)
  - funct: código de função (extensão do opcode)

## Instruções: A Linguagem de Máquina

### ► Formato R - Exemplo

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

add \$t0, \$s1, \$s2

special	\$s1	\$s2	\$t0	0	add
---------	------	------	------	---	-----

0	17	18	8	0	32
---	----	----	---	---	----

000000	10001	10010	01000	00000	100000
--------	-------	-------	-------	-------	--------

000000|0001|100100|000000000|000000<sub>2</sub> = 02324020<sub>16</sub>

## Instruções: A Linguagem de Máquina

### ► Representação em Hexadecimal

Representação compacta de strings de bits

4 bits por dígito hexadecimal

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

Exemplo: eca8 6420

1110 1100 1010 1000 0110 0100 0010 0000

## Instruções: A Linguagem de Máquina

### ► MIPS - Formato I



Formato de instrução utilizado pelas instruções imediatas e de transferência de dados.

*Princípio de Projeto IV : um bom projeto exige bons compromissos.*

## Instruções: A Linguagem de Máquina

### ► Operações Lógicas

Operation	C	Java	MIPS
Shift à esquerda	<<	<<	sll
Shift à direita	>>	>>>	srl
AND bit a bit	&	&	and, andi
OR bit a bit			or, ori
NOT bit a bit	~	~	nor



## Instruções: A Linguagem de Máquina

### ► Operações de deslocamento

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

**shamt:** quantas posições de deslocamento (shift amount)

#### **Deslocamento lógico à esquerda**

deslocamento à esquerda e preenchimento com 0(s)

deslocamento por  $i$  bits multiplica por  $2^i$

#### **Deslocamento lógico à direita**

deslocamento à direita e preenchimento com 0(s)

deslocamento à direita por  $i$  bits divide por  $2^i$  (para números sem sinal)

## Instruções: A Linguagem de Máquina

### ► Operação AND

Útil para mascarar bits em uma word

and \$t0, \$t1, \$t2

\$t2	0000 0000 0000 0000 0000	1101	1100 0000
\$t1	0000 0000 0000 0000 0011	1100	0000 0000
\$t0	0000 0000 0000 0000 0000	1100	0000 0000

## Instruções: A Linguagem de Máquina

### ► Operação OR

or \$t0, \$t1, \$t2

\$t2	0000 0000 0000 0000 0000	1101	1100 0000
\$t1	0000 0000 0000 0000 0011	1100	0000 0000
\$t0	0000 0000 0000 0000 0011	1101	1100 0000

## Instruções: A Linguagem de Máquina

### ► Operação NOT

nor \$t0, \$t1, \$zero

\$t1

0000 0000 0000 0000 0011 1100 0000 0000

\$t0

1111 1111 1111 1111 1100 0011 1111 1111

## Instruções: A Linguagem de Máquina

### ► Operações Condicionais

beq (branch if equal)

beq registrador1, registrador2, L1

bne (branch if not equal)

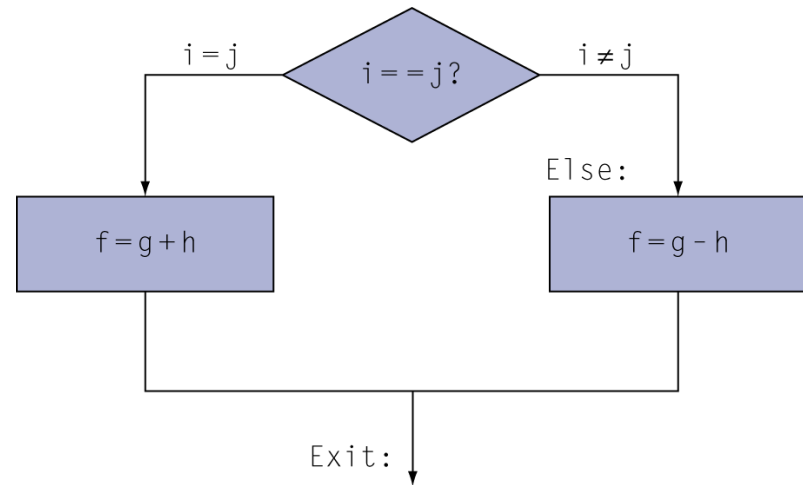
beq registrador1, registrador2, L1

## Instruções: A Linguagem de Máquina

### ► Operações Condicionais

Código C

```
if (i==j) f = g+h;  
else f = g-h;
```



Supondo que as cinco variáveis de  $f$  a  $j$  correspondem aos cinco registradores  $\$s0$  a  $\$s4$  qual é o código compilado para a instrução `if` em C?

## Instruções: A Linguagem de Máquina

### ► Operações Condicionais

Código MIP compilado:

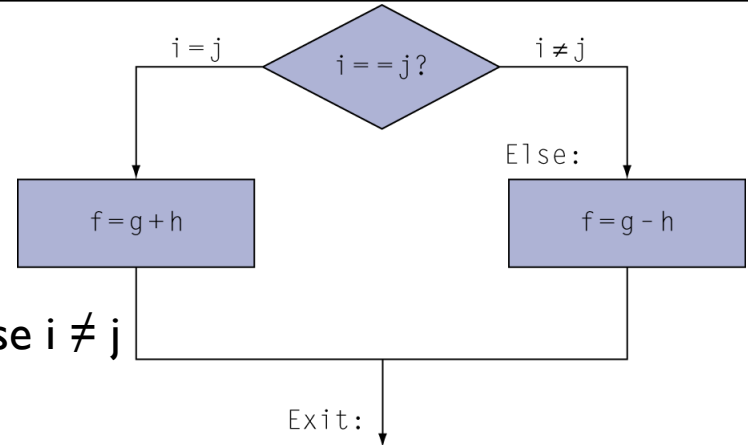
```
bne $s3,$s4,Else    # vá para Else se  $i \neq j$ 
```

```
add $s0,$s1,$s2    #  $f = g+h$ 
```

```
j Exit #vá para Exit
```

```
Else:  sub $s0,$s1,$s2    #  $f = g-h$ 
```

```
Exit:
```



## Instruções: A Linguagem de Máquina

### ► Compilando um loop While em C

Código C

```
while (save[i] == k) i += 1;
```

Supondo que `i` está associado a `$s3`, `k` a `$s5` e que o endereço de `save` está em `$s6` qual é o código compilado para a instrução `while` em C?



## Instruções: A Linguagem de Máquina

### ► Compilando um loop While em C

Código MIP compilado:

```
Loop: sll  $t1, $s3, 2          # Reg. temp. $t1 = 4*i
      add  $t1, $t1, $s6       # $t1 = endereço de save[i]
      lw   $t0, 0($t1)         # Reg. temp. $t0 = save[i]
      bne  $t0, $s5, Exit      # vá para Exit se save[i] ≠ k
      addi $s3, $s3, 1         # i=i+1
      j    Loop               # vá para Loop
Exit: ...
```

## Instruções: A Linguagem de Máquina

### ► Resumo

Nome	Exemplo	Comentários
32 registradores	\$s0, \$s1, ..., \$s7 \$t0, \$t1, ..., \$t7	Locais rápidos para dados. No MIPS, os dados precisam estar em registradores para a realização de operações aritméticas. Os registradores \$s0-\$s7 são mapeados para 16-23; \$t0-\$t7 são mapeados para 8-15. O registrador MIPS \$zero é sempre igual a 0.

## Instruções: A Linguagem de Máquina

### ► Resumo

Nome	Exemplo	Comentários
$2^{30}$ words na memória	Memória[0], Memória[4]..... Memória[4294967292]	Acessadas apenas por instruções de transferência de dados no MIPS. O MIPS utiliza endereços em bytes, de modo que os endereços em words sequenciais diferem em 4 vezes. A memória contém estruturas de dados, arrays e spilled registers.

## Instruções: A Linguagem de Máquina

### ► Resumo

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	Add	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	Três operandos; dados no registrador
	subtract	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	Três operandos; dados no registrador

## Instruções: A Linguagem de Máquina

### ► Resumo

Categoria	Instrução	Exemplo	Significado	Comentários
Transferência de dados	load word	lw \$s1,100(\$s2)	$\$s1 = \text{Memória}[\$s2 + 100]$	Dados da memória para o registrador
	store word	sw \$s1,100(\$s2)	$\text{Memória}[\$s2 + 100] = \$s1$	Dados do registrador para memória

## Instruções: A Linguagem de Máquina

### ► Resumo

Categoria	Instrução	Exemplo	Significado	Comentários
Lógica	and	and \$s1,\$s2,\$s3	$\$s1 = \$s2 \& \$s3$	Três operadores em registrador; AND bit a bit
	or	or \$s1,\$s2,\$s3	$\$s1 = \$s2 \mid \$s3$	Três operadores em registrador; OR bit a bit
	nor	nor \$s1,\$s2,\$s3	$\$s1 = \sim(\$s2 \mid \$s3)$	Três operadores em registrador; NOR bit a bit
	and immediate	andi \$s1,\$s2,100	$\$s1 = \$s2 \& 100$	AND bit a bit entre registrador com constante
	or immediate	ori \$s1,\$s2,100	$\$s1 = \$s2 \mid 100$	OR bit a bit entre registrador com constante
	shift left logical	sll \$s1,\$s2,10	$\$s1 = \$s2 \ll 10$	Deslocamento à esquerda por constante
	shift right logical	srl \$s1,\$s2,10	$\$s1 = \$s2 \gg 10$	Deslocamento à direita por constante

## Instruções: A Linguagem de Máquina

### ► Resumo

Categoria	Instrução	Exemplo	Significado	Comentários
Desvio condicional	branch on equal	beq \$s1,\$s2,L	If (\$s1 == \$s2) go to L	Testa igualdade e desvia
	branch on not equal	bne \$s1,\$s2,L	If (\$s1 != \$s2) go to L	Testa desigualdade e desvia
	set on less than	slt \$s1,\$s2,\$s3	If (\$s2 < \$s3) \$s1=1; Else \$s1=0	Compara menor que; usado com beq, bne
	set on less than immediate	slti \$s1,\$s2,100	If (\$s2 < 100) \$s1=1; Else \$s1=0	Compara menor que imediato; usado com beq, bne

## Instruções: A Linguagem de Máquina

### ► Resumo

Categoria	Instrução	Exemplo	Significado	Comentários
Desvio incondicional	jump	j L	go to L	Desvia para endereço de destino



## Instruções: A Linguagem de Máquina

### ► Resumo

Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções MIPS de 32 bits
Formato R	R	op	rs	rt	rd	shamt	funcnt	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato para transferência de dados e desvios

- op: operação básica da instrução, tradicionalmente chamado de **opcode**.
- rs: o registrador do primeiro operando de origem.
- rt: o registrador do segundo operando de origem.
- rd: o registrador do operando de destino
- shamt: "shift amount" (quantidade de deslocamento)
- funcnt: função. Esse campo seleciona a variante específica da operação no campo op e , às vezes, é chamado de **código de função**.

## Instruções: A Linguagem de Máquina

### ► Resumo

Nome	Formato	Exemplo						Comentários
add	R	0	18	19	17	0	32	add \$s1,\$s2,\$s3
sub	R	0	18	19	17	0	34	sub \$s1,\$s2,\$s3
lw	I	35	18	17	100			lw \$s1,100(\$s2)
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções MIPS de 32 bits
Formato R	R	op	rs	rt	rd	shamt	funcnt	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato para transferência de dados e desvios

## Instruções: A Linguagem de Máquina

### ► Resumo

Nome	Formato	Exemplo						Comentários
sw	I	43	18	17	100			sw \$s1,100(\$s2)
and	R	0	18	19	17	0	36	and \$s1,\$s2,\$s3
or	R	0	18	19	17	0	37	or \$s1,\$s2,\$s3
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções MIPS de 32 bits
Formato R	R	op	rs	rt	rd	shamt	fucnt	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato para transferência de dados e desvios

## Instruções: A Linguagem de Máquina

### ► Resumo

Nome	Formato	Exemplo						Comentários
nor	R	0	18	19	17	0	39	nor \$s1,\$s2,\$s3
andi	I	12	18	17	100			andi \$s1,\$s2,100
ori	I	13	18	17	100			ori \$s1,\$s2,100
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções MIPS de 32 bits
Formato R	R	op	rs	rt	rd	shamt	fucnt	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato para transferência de dados e desvios

## Instruções: A Linguagem de Máquina

### ► Resumo

Nome	Formato	Exemplo						Comentários
sll	R	0	0	18	17	10	0	sll \$s1,\$s2,10
srl	R	0	0	18	17	10	2	srl \$s1,\$s2,10
beq	I	4	17	18	25			beq \$s1,\$s2,100
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções MIPS de 32 bits
Formato R	R	op	rs	rt	rd	shamt	funcnt	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato para transferência de dados e desvios

## Instruções: A Linguagem de Máquina

### ► Resumo

Nome	Formato	Exemplo						Comentários
bne	I	5	17	18	25			bne \$s1,\$s2,100
slt	R	0	18	19	17	0	42	slt \$s1,\$s2,\$s3
j	J	2	2500					J 10000
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções MIPS de 32 bits
Formato R	R	op	rs	rt	rd	shamt	fucnt	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato para transferência de dados e desvios

## Instruções: A Linguagem de Máquina

### ► Exercício

Implementar em código MIPS

$$S = \sum_{k=1}^{20} (4k + 2)$$

## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

Etapas na execução de um procedimento (visto pelo programa em execução):

- Colocar parâmetros em um lugar onde o procedimento possa acessá-los.
- Transferir o controle para o procedimento.
- Adquirir os recursos de armazenamento necessários para o procedimento.
- Realizar a tarefa desejada.
- Colocar o valor de retorno em um local onde o programa que o chamou possa acessá-lo.



## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

O MIPS utiliza a seguinte convenção na alocação de seus 32 registradores para chamada de procedimento:

- \$a0-\$a3: quatro registradores de argumento, para passar parâmetros
- \$v0-\$v1: dois registradores de valor, para valores de retorno.
- \$ra: um registrador de endereço de retorno, para retornar ao ponto de origem.

## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

#### Instrução **jump-and-link (jal)**

A instrução jal desvia para um endereço e simultaneamente salva o endereço da instrução seguinte (**endereço de retorno**) no registrador \$ra.

```
jal EndereçoProcedimento
```

A instrução jal salva o PC+4 no registrador \$ra

#### Instrução **jump register (jr)**

A instrução faz um desvio incondicional para o endereço especificado em um registrador.

## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

- programa que chama um procedimento (**caller**) coloca os valores de parâmetro em \$a0-\$a3 e utiliza jal X para desviar para o procedimento X (às vezes chamado **calee**).
- procedimento (calee) realiza os cálculos, coloca os resultados em \$v0-\$v1 e retorna o controle para caller usando jr \$ra.

### Usando mais registradores

Se precisarmos utilizar mais registradores para um procedimento do que os quatro registradores para argumentos e os dois para valores de retorno podemos utilizar os **spilled registers** em memória.

*O processo de colocar variáveis menos utilizadas (ou aquelas que serão utilizadas mais tarde) na memória é denominado **spilling registers**.*

## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

#### Compilando um procedimento em C que não chama outro procedimento

```
int exemplo_folha(int g, int h, int i, int j)
{
    int f;
    f=(g+h) - (i+j);
    return f;
}
```

As variáveis de parâmetro g, h, i e j correspondem aos registradores \$a0, \$a1, \$a2 e \$a3, e f corresponde a \$s0 (desta forma devemos salvar \$s0 na pilha). Resultado em \$v0.

# Instruções: A Linguagem de Máquina

## ► Suporte a procedimentos no hardware do computador

### Código MIPS

exemplo\_folha:

addi \$sp, \$sp, -12 # ajusta a pilha criando espaço para três itens

sw \$t1, 8(\$sp) # salva reg. \$t1 para usar depois

sw \$t0, 4(\$sp) # salva reg. \$t0 para usar depois

sw \$s0, 0(\$sp) # salva reg. \$s0 para usar depois

add \$t0, \$a0, \$a1 # reg. \$t0 contém (g+h)

add \$t1, \$a2, \$a3 # reg. \$t1 contém (i+j)

sub \$s0, \$t0, \$t1 # reg. \$s0 recebe (g+h)-(i+j)

add \$v0, \$s0, \$zero # \$v0 = \$s0 + 0

lw \$s0, 0(\$sp) # restaura reg. \$s0 para caller

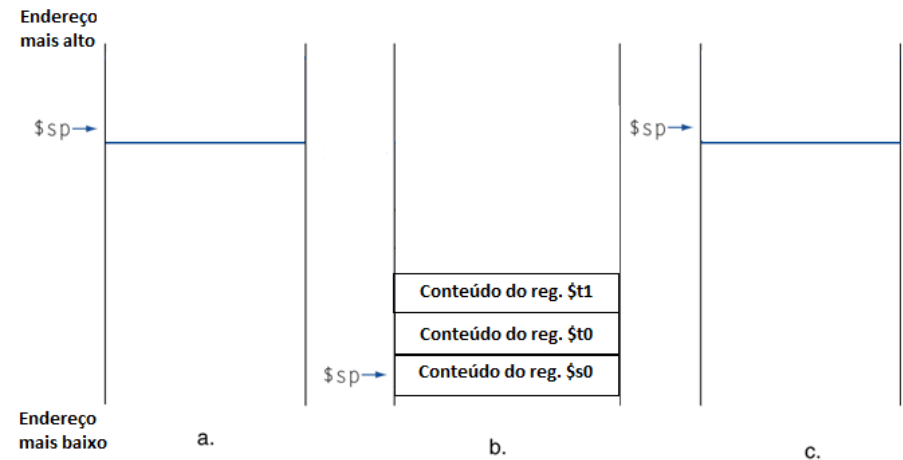
lw \$t0, 4(\$sp) # restaura reg. \$t0 para caller

lw \$t1, 8(\$sp) # restaura reg. \$t1 para caller

addi \$sp, \$sp, 12 # ajusta pilha para excluir três itens

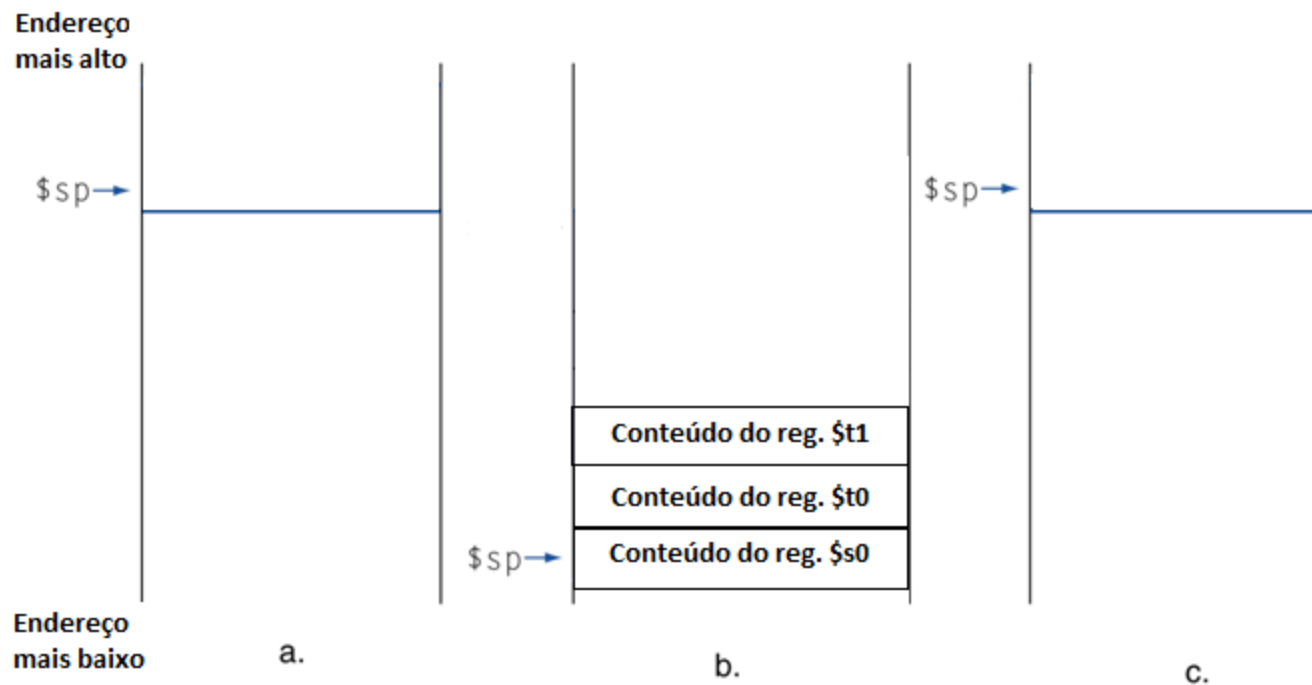
jr \$ra # desvia de volta à rotina que chamou

```
int exemplo_folha(int g, int h, int i, int j)
{
    int f;
    f=(g+h) - (i+j);
    return f;
}
```



## Instruções: A Linguagem de Máquina

- Suporte a procedimentos no hardware do computador



## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

O MIPS separa 18 dos registradores em dois grupos:

\$t0 a \$t9: 10 registradores temporários que não são preservados pelo procedimento chamado em uma chamada de procedimento

\$s0 a \$s7: 8 registradores salvos **que precisam ser preservados em uma chamada de procedimento** (se forem usados, o procedimento chamado os salva e restaura)

## Instruções: A Linguagem de Máquina

- Suporte a procedimentos no hardware do computador
- Código MIPS**

leaf_example:			
addi	\$sp,	\$sp, -4	Save \$s0 on stack
sw	\$s0,	0(\$sp)	
add	\$t0,	\$a0, \$a1	Procedure body
add	\$t1,	\$a2, \$a3	
sub	\$s0,	\$t0, \$t1	
add	\$v0,	\$s0, \$zero	Result
lw	\$s0,	0(\$sp)	Restore \$s0
addi	\$sp,	\$sp, 4	
jr	\$ra		Return



## Instruções: A Linguagem de Máquina

### Exercício I

Implementar no código MIPS o procedimento dado em C:

```
int calculo (int x, int y, int z, int w)
{
    int f;
    f=[(x+y) *(z+w)] – 5
    return f;
}
```

## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

#### Procedimentos aninhados

Caller: empilha quaisquer registradores de argumento (\$a0 - \$a3) ou registradores temporários (\$t0 - \$t9) que sejam necessários após a chamada.

Callee: empilha o registrador de endereço de retorno \$ra e quaisquer registradores salvos (\$s0 - \$s7) usados por ele.

O stack pointer \$sp é ajustado para levar em consideração a quantidade de registradores colocados na pilha. No retorno, os registradores são restaurados memória e o stack pointer é ajustado.

## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

#### Procedimentos aninhados

C code:

```
int fact (int n)
{
    if (n < 1) return f;
    else return n * fact(n - 1);
}
```

- Argumento n em \$a0
- Resultado em \$v0

## Instruções: A Linguagem de Máquina

### Exercício 2

Implementar no código MIPS o procedimento dado em C:

```
int fact (int n)
{
    if (n < 1) return f;
    else return n * fact(n - 1);
}
```

slt \$s1,\$s2,100	If (\$s2 < 100) \$s1=1; Else \$s1=0
-------------------	--

## Instruções: A Linguagem de Máquina

### ► Suporte a procedimentos no hardware do computador

#### Procedimentos aninhados

fact:

```
addi $sp,$sp,-8 # ajusta a pilha para 2 itens
sw $ra, 4($sp) # salva o endereço de retorno
sw $a0,0($sp) # salva o argumento n
slti $t0,$a0,1 # teste par n < 1
beq $t0,$zero,LI # se n>=1, vai para LI
    addi $v0,$zero,1 # retorna 1
    addi $sp,$sp,8 # retira 2 itens da pilha
    jr $ra # retorna para depois de jal
```

```
LI: addi $a0,$a0,-1 # n>=1 : argumento recebe (n-1)
    jal fact # chama fact com (n-1)
    lw $a0,0($sp) # retorna de jal: restaura o argumento n
    lw $ra,4($sp) # restaura o endereço de retorno
    addi $sp,$sp,8 # ajusta stack pointer para retirar 2 itens
    mul $v0,$a0,$v0 # retorna n*fact(n-1)
    jr $ra #retorna para o procedimento que chamou
```

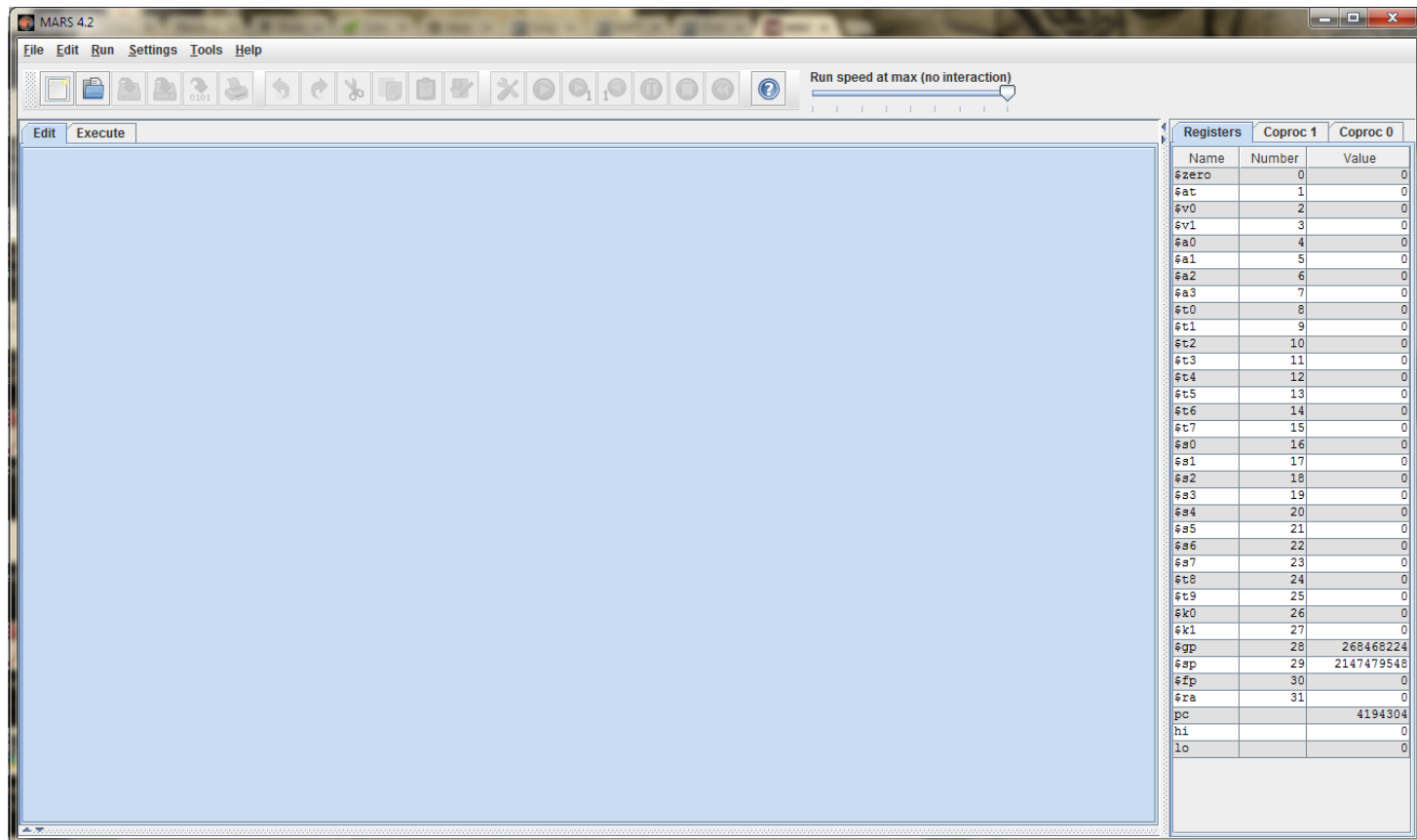
```
int fact (int n)
{
    if (n < 1) return f;
    else return n * fact(n - 1);
}
```

slti \$s1,\$s2,100	If (\$s2 < 100) \$s1=1; Else \$s1=0
--------------------	--

## Instruções: A Linguagem de Máquina

### ► MARS (MIPS Assembler and Runtime Simulator)

O software pode ser baixado no link: <http://courses.missouristate.edu/kenvollmar/mars/>



## Instruções: A Linguagem de Máquina

### ► Subconjunto de diretivas definidas para o montador MIPS

**Para controle de segmento:**

**.data <address>**

Os valores definidos a seguir devem ser colocados no segmento de dados do utilizador, opcionalmente a partir do endereço <address>.

**.text <address>**

Os valores definidos a seguir devem ser colocados no segmento de texto do utilizador, opcionalmente a partir do endereço <address>. Todos os valores devem ter 32 bits, ou seja, serem instruções ou palavras (words).

**.kdata <address>**

Os valores definidos a seguir devem ser colocados no segmento de dados do kernel, opcionalmente a partir do endereço <address>.

## Instruções: A Linguagem de Máquina

### ► Subconjunto de diretivas definidas para o montador MIPS

**Para controle de segmento:**

**.ktext <address>**

Os valores definidos a seguir devem ser colocados no segmento de texto do kernel, opcionalmente a partir do endereço <address>.

**Para definição de constantes e variáveis na memória:**

**.ascii str**

armazena uma *string* em memória sem lhe acrescentar o terminador NULL.

**.asciiz str**

armazena uma *string* em memória acrescentando-lhe o terminador NULL.



## Instruções: A Linguagem de Máquina

### ► Subconjunto de diretivas definidas para o montador MIPS

Para definição de constantes e variáveis na memória:

**.byte  $b_1, \dots, b_n$**

armazena as grandezas de 8 bits  $b_1, \dots, b_n$  em sucessivos bytes de memória.

**.half  $h_1, \dots, h_n$**

armazena as grandezas de 16 bits  $h_1, \dots, h_n$  em sucessivas meias palavras de memória.

**.word  $w_1, \dots, w_n$**

armazena as grandezas de 32 bits  $w_1, \dots, w_n$  em sucessivas palavras de memória.

**.float  $f_1, \dots, f_n$**

armazena os números em vírgula flutuante com precisão simples (32 bits)

$f_1, \dots, f_n$  em posições de memória sucessivas.

## Instruções: A Linguagem de Máquina

- ▶ **Subconjunto de diretivas definidas para o montador MIPS**

**Para definição de constantes e variáveis na memória:**

**.double d1, ..., dn**

armazena os números em vírgula flutuante com precisão dupla (64 bits) **d1, ..., dn** em posições de memória sucessivas.

**.space n**

reserva **n bytes**

## Instruções: A Linguagem de Máquina

### ► Subconjunto de diretivas definidas para o montador MIPS

Para controle do alinhamento:

**.align n**

alinha o próximo item num endereço múltiplo de  $2^n$ . Por exemplo **.align 2** seguido de **.word xpto** garante que a palavra **xpto** é armazenada num endereço múltiplo de 4.

**.align 0**

desliga o alinhamento automático das diretivas **.half**, **.word**, **.float**, e **.double** até à próxima diretiva **.data** ou **.kdata**.

Para referências externas:

**.globl sym**

declara que o símbolo **sym** é global e pode ser referenciado a partir de outros arquivos.

## Instruções: A Linguagem de Máquina

### ► Subconjunto de diretivas definidas para o montador MIPS

Para referências externas:

#### **.extern sym size**

declara que o item associado a **sym** ocupa **size** bytes e é um símbolo global. Esta diretiva permite ao montador armazenar o item numa porção do segmento de dados que seja eficientemente acedido através do registo \$gp.

### **Chamadas ao Sistema (*System Calls*)**

O SPIM dispõe de um conjunto de funções típicas de um sistema operacional.

## Instruções: A Linguagem de Máquina

### ► Chamadas ao Sistema (*System Calls*)

O MARS dispõe de um conjunto de funções típicas de um sistema operacional.

Função	\$v0	Parâmetros	Retorno
print_int (int)	1	\$a0 = int	
print_float (float)	2	\$f12 = float	
print_double (double)	3	\$f12 = double	
print_string (string)	4	\$a0 = string	
read_int (int)	5		\$v0 = integer
read_float (float)	6		\$f0 = float
read_double (double)	7		\$f0 = double
read_string (string)	8	\$a0 = buffer, \$a1 = lenght	
sbrk	9	\$a0 = amount	\$v0 = address
exit	10		

## Instruções: A Linguagem de Máquina

### ► Chamadas ao Sistema (*System Calls*)

A utilização das chamadas ao sistema é feita por meio da instrução de chamada ao sistema **syscall**.

Para fazer uma chamada ao sistema devemos carregar o código da chamada no registro **\$v0** e os argumentos nos registradores **\$a0** e **\$a1** (ou **\$f12** para valores em ponto flutuante). O valor de retorno (caso exista), é carregado no registro **\$v0** (ou **\$f0** no caso de ponto flutuante)

Exemplo:

```
.data
str:  .ascii "Arquitetura e Organização de Computadores!"
.text
.globl main
main: li $v0, 4
      la $a0, str
      syscall # print_string ("Arquitetura e Organização de Computadores!")
```

## Instruções: A Linguagem de Máquina

### ► Registradores da CPU

- 32 registradores de uso geral
- registradores de multiplicação/divisão (HI/LO)
- contador de programa (PC)

Nome lógico	Nome real	Utilização
\$zero	\$0	Constante zero
\$at	\$1	Reservado pelo assembler
\$v0 e \$v1	\$2 e \$3	Cálculo de expressões e valor de retorno das funções
\$a0....\$a3	\$4....\$7	Primeiros 4 parâmetros das funções
\$t0....\$t7	\$8....\$15	Geral (pode não ser preservado pelas funções)
\$s0....\$s7	\$16....\$23	Geral (deve ser preservado pelas funções)
\$t8 e \$t9	\$24 e \$25	Geral (pode não ser preservado pelas funções)
\$k0 e \$k1	\$26 e \$27	Reservado pelo kernel do S.O
\$gp	\$28	Ponteiro para área global
\$sp	\$29	<i>Stack Pointer</i>
\$fp	\$30	<i>Frame Pointer</i>
\$ra	\$31	Endereço de retorno das funções ( <i>Return Address</i> )

## Instruções: A Linguagem de Máquina

### ► Exercício 1)

Elaborar um programa para carregar o valor 3 no registrador \$t0, o valor 4 no registrador \$t1. Some os dois valores e multiplique por 4 (resultado em \$t3). Utilize a chamada de sistema exit para finalizar.



## Instruções: A Linguagem de Máquina

### ► Exercício 1)

```
# exemplo1.s
.text
main:
    addiu $t0, $0, 3
    addiu $t1, $0, 4
    addu $t2, $t0, $t1
    sll $t3, $t2, 2
# Saída
    addiu $v0, $0, 10
    syscall
```

## Instruções: A Linguagem de Máquina

- ▶ **Exercício 2)** Elaborar um programa em **MIPS** que tenha com entrada o número de avaliações de uma disciplina e forneça como saída a média das notas.

## Instruções: A Linguagem de Máquina

### ► Solução – Exercício 2

```
.data
msg1: .asciiz "\nEntre com o número de avaliações da disciplina: "
msg2: .asciiz "\nEntre um valor para a nota "
msg3: .asciiz ": "
msg4: .asciiz "\nA média das notas é: "
.text
.globl main
main:
add $t0, $zero, $zero # Limpa o conteúdo de $t0
add $t1, $zero, $zero # Limpa o conteúdo de $t1
numnotas:
li $v0, 4    # Codigo SysCall p/ escrever strings
la $a0, msg1    # Parâmetro (string a ser escrita)
syscall
li $v0, 5    # Codigo SysCall p/ ler inteiros
syscall      # Inteiro lido vai ficar em $v0
add $s0, $v0, $zero    # Armazena em $s0 o número de notas
```

## Instruções: A Linguagem de Máquina

```

loop_notas:
addi $t0, $t0, 1          # Incrementa $t0 - contador de notas
li $v0, 4                 # Código SysCall p/ escrever strings
la $a0, msg2              # Parâmetro (string a ser escrita)
syscall
li $v0, 1                 # Código SysCall p/ escrever inteiros
add $a0, $zero, $t0       # Parâmetro (inteiro a ser escrito)
syscall
li $v0, 4                 # Código SysCall p/ escrever strings
la $a0, msg3              # Parâmetro (string a ser escrita)
syscall
li $v0, 5                 # Código SysCall p/ ler inteiros
syscall                   # Inteiro lido vai ficar em v0
add $t1, $t1, $v0         # Soma a nota ao total
bne $t0, $s0, loop_notas  # Enquanto não preencher todas as notas, loop
Calcula:
div $t1, $s0              # Divide o total pelo numero de notas
mflo $t2                  # Move o resultado para $t2
li $v0, 4                 # Código SysCall p/ escrever strings
la $a0, msg4              # Parametro (string a ser escrita)
syscall
li $v0, 1                 # Código SysCall p/ escrever inteiros
add $a0, $zero, $t2       # Parâmetro (inteiro a ser escrito)
syscall
li $v0, 5                 # Apenas para esperar um [ENTER]
syscall

```

div registrador1, registrador2  
 - divide o registrador1 pelo registrador2 e guarda o resultado nos registradores especiais Hi e Lo  
 Obs: O quociente fica guardado em Lo e o resto fica guardado em Hi

## Instruções: A Linguagem de Máquina

### ► Exemplos

```
Loop: r = r + A[i]  
      i=i+1  
      if (i!=h) go to Loop;
```

## Instruções: A Linguagem de Máquina

**Solução:** supondo que as variáveis *r*, *i* e *h* estejam associadas aos registradores *\$s1*, *\$s2* e *\$s3* e que o endereço base do *array A* esteja no registrador *\$s4*

```
Loop: add $t1, $s2, $s2    # $t1 = 2*i
      add $t1, $t1, $t1    # $t1 = 4*i
      add $t1, $t1, $s4    # $t1 recebe o endereço de A[i]
      lw $t0, 0($t1)      # $t0 recebe A[i]
      add $s1, $s1, $t0    # r = r + A[i]
      add $t1, $t1, $zero  # i = i + 1
      bne $s2, $s3, Loop  # se i != h salta para Loop
```

```
Loop: r = r + A[i]
      i = i + 1
      if (i != h) go to Loop;
```

## Instruções: A Linguagem de Máquina

### ► Exemplos

```
while ( save[i] == k)  
    i = i + j;
```

## Instruções: A Linguagem de Máquina

### Solução:

Suponha que *i*, *j* e *k* correspondam aos registradores \$s3, \$s4 e \$s5, e que o endereço inicial do array *save* esteja armazenado em \$s6.

```
Loop: add $t1, $s3, $s3
      add $t1, $t1, $t1
      add $t1, $t1, $s6
      lw  $t0, 0 ( $t1)
      bne $t0, $s5, Exit
      add $s3, $s3, $s4
      j   Loop
```

Exit:

```
while ( save[i] == k)
    i = i + j;
```



## Instruções: A Linguagem de Máquina

### ► Lista de Exercícios

(a)

```
i = 0;  
enquanto i < 100 faça  
    x = x + V[i];  
fim-enquanto
```

## Instruções: A Linguagem de Máquina

### Solução:

Suponha que *i* e *x* correspondam aos registradores *\$s4* e *\$s2* e que o endereço inicial do array *V* esteja armazenado em *\$s3*.

```
repita:  add $s4,$zero, $zero
        addi $t1,$zero,100
        slt $s1,$s4,$t1
        beq $s1,$zero, fim
        add $t0,$s4,$s4
        add $t0,$t0,$t0
        add $t0,$t0,$s3
        lw $t2,0($t0)
        add $s2,$s2,$t2
        addi $s4,$s4,1
        j repita
fim:
```

```
i = 0;
enquanto i < 100 faça
    x = x + V[i];
fim-enquanto
```

## Instruções: A Linguagem de Máquina

### ► Exemplo de Leitura e Escrita de um vetor de inteiros

```
1  .data
2  ent: .asciiz "Insira um valor de Vet["
3  ent2: .asciiz "]: "
4  .align 2
5  vet: .space 20
6
7
8  .text
9  main: la $a0, vet # Endereço do vetor como parâmetro
10         jal leitura # leitura(vet)
11         move $a0, $v0 # Endereço do vetor retornado
12         jal escrita # escrita(vet)
13         li $v0, 10 # Código para finalizar o programa
14         syscall # Finaliza o programa
```

## Instruções: A Linguagem de Máquina

### ► Exemplo de Leitura e Escrita de um vetor de inteiros

```
16 leitura:
17     move $t0, $a0 # Salva o endereço base de vet
18     move $t1, $t0 # Endereço de vet[i]
19     li $t2, 0 # i=0
20 1:   la $a0, ent # Carrega endereço da string
21     li $v0, 4 # Código de impressão de string
22     syscall # Impressão de string
23     move $a0, $t2 # Carrega o índice do vetor
24     li $v0, 1 # Código de impressão de inteiro
25     syscall # Imprime o índice i
26     la $a0, ent2 # Carrega o endereço da string
27     li $v0, 4 # Código de impressão de string
28     syscall # Impressão da string
29     li $v0, 5 # Código de leitura de inteiro
30     syscall # Leitura do valor
31     sw $v0, ($t1) # Salva o valor lido em vet[i]
32     add $t1, $t1, 4 # Endereço de vet[i+1]
33     addi $t2, $t2, 1 # i++
34     blt $t2, 5, 1 # if(i < 5) goto 1
35     move $v0, $t0 # Endereço de vet para retorno
36     jr $ra # Retorna para a main
37
```

## Instruções: A Linguagem de Máquina

### ► Exemplo de Leitura e Escrita de um vetor de inteiros

```
38  escrita:
39      move $t0, $a0 # Salva o endereço base de vet
40      move $t1, $t0 # Endereço de vet[i]
41      li $t2, 0 # i=0
42  e:   lw $a0, ($t1) # Carrega o valor de Vet[i]
43      li $v0, 1 # Código de impressão de inteiro
44      syscall # Imprime vet[i]
45      li $a0, 32 # Código ASCII para espaço
46      li $v0, 11 # Código de impressão de caractere
47      syscall # Imprime um espaço
48      add $t1, $t1, 4 # Endereço de vet[i+1]
49      addi $t2, $t2, 1 # i++
50      blt $t2, 5, e # if(i < 5) goto e
51      move $v0, $t0 # Endereço de vet para retorno
52      jr $ra # Retorno para a main
```

## Instruções: A Linguagem de Máquina

### ► Exemplo de Leitura e Escrita de um vetor de inteiros

```
Insira um valor de Vet[0]: 1
Insira um valor de Vet[1]: 2
Insira um valor de Vet[2]: 3
Insira um valor de Vet[3]: 4
Insira um valor de Vet[4]: 5
1 2 3 4 5
-- program is finished running --
```

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

- ❑ Floating Point Unit (FPU) ou unidade de ponto flutuante
- ❑ Arquitetura MIPS32: a FPU é implementada principalmente por meio do coprocessador 1 (coprocessador de ponto flutuante), também denominado de floating-point accelerator (FPA).

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

- ❑ Representação binária: padrão IEEE 754 – o IEEE *for Binary Floating-Point Arithmetic*.
- ❑ A FPU do MIPS oferece suporte aos tipos de dados em ponto flutuante e aos tipos de dados em ponto fixo.
- ❑ Tipos de dados em ponto flutuante: *single precision* e *double precision*

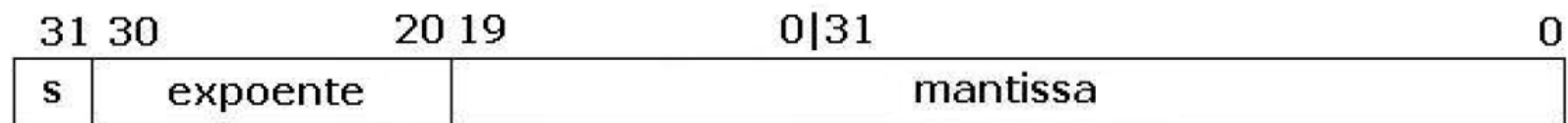
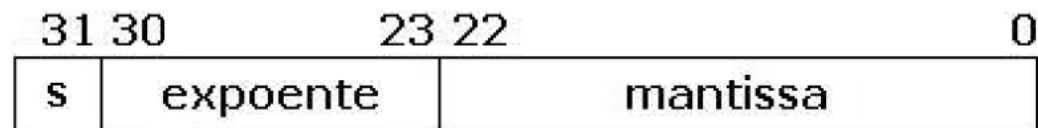


## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

□ Formatos definidos:

- ponto fixo 32-bit
- ponto flutuante 32-bit single precision (tipo S)
- ponto flutuante 64-bit double precision (tipo D)



## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### ❑ Registradores

- 32 registradores de ponto flutuante: \$f0 - \$f31
- Double precision: a CPU realiza operações aritméticas nos 16 registradores pares, sendo os 16 registradores ímpares utilizados para cuidar dos 32 bits mais significativos dos números de 64 bits.

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

❑ Registradores (convenção para registradores da FPU)

Registrador	Nome	Uso
\$f0,\$f2	fv0-fv1	Valor retornado por funções.
\$f4,\$f6,\$f8,\$f10	ft0-ft3	Temporários.
\$f12,\$f14	fa0-fa1	Argumentos de funções
\$f16,\$f18	ft4-ft5	Temporários.
\$f20,\$f22,\$f24 \$f26,\$f28,\$f30	fs0-fs5	Variáveis de registro.

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### ❑ Instruções

- As instruções da FPU possuem formatos diferentes dos especificados para as instruções da CPU (possuem normalmente **uma versão single** e uma **versão double**).

#### ❑ Instruções Single Precision

**l.s \$fd, addr**

Carrega 32 bits de dados a partir do endereço addr em um registrador ponto flutuante \$fd (onde d é 0, 1, 2,...,31).

## Instruções: A Linguagem de Máquina

### ► MARS (Convenções)

#### *Operand Key for Example Instructions*

label, target	any textual label
\$t1, \$t2, \$t3	any integer register
\$f2, \$f4, \$f6	<i>even-numbered</i> floating point register
\$f0, \$f1, \$f3	any floating point register
\$8	any Coprocessor 0 register
1	condition flag (0 to 7)
10	unsigned 5-bit integer (0 to 31)
-100	signed 16-bit integer (-32768 to 32767)
100	unsigned 16-bit integer (0 to 65535)
100000	signed 32-bit integer (-2147483648 to 2147483647)

#### *Load & Store addressing mode, basic instructions*

-100 (\$t2)	sign-extended 16-bit integer added to contents of \$t2
-------------	--

## Instruções: A Linguagem de Máquina

### ► MARS (Convenções)

#### *Load & Store addressing modes, pseudo instructions*

(\$t2)	contents of \$t2
-100	signed 16-bit integer
100	unsigned 16-bit integer
100000	signed 32-bit integer
100 (\$t2)	zero-extended unsigned 16-bit integer added to contents of \$t2
100000 (\$t2)	signed 32-bit integer added to contents of \$t2
label	32-bit address of label
label (\$t2)	32-bit address of label added to contents of \$t2
label+100000	32-bit integer added to label's address
label+100000 (\$t2)	sum of 32-bit integer, label's address, and contents of \$t2

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### ☐ Instruções (MARS)

☐ l.s \$f1,(\$t2)

☐ l.s \$f1,-100

☐ l.s \$f1,100000

☐ l.s \$f1,100000(\$t2)

☐ l.s \$f1,label

☐ l.s \$f1,label(\$t2)

☐ l.s \$f1,label+100000

☐ l.s \$f1,label+100000(\$t2)

#### *Load & Store addressing modes, pseudo instructions*

(\$t2)	contents of \$t2
-100	signed 16-bit integer
100	unsigned 16-bit integer
100000	signed 32-bit integer
100 (\$t2)	zero-extended unsigned 16-bit integer added to contents of \$t2
100000 (\$t2)	signed 32-bit integer added to contents of \$t2
label	32-bit address of label
label (\$t2)	32-bit address of label added to contents of \$t2
label+100000	32-bit integer added to label's address
label+100000 (\$t2)	sum of 32-bit integer, label's address, and contents of \$t2

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### ☐ Instruções (MARS)

☐ s.s \$f1,(\$t2)

☐ s.s \$f1,-100

☐ s.s \$f1,100000

☐ s.s \$f1,100000(\$t2)

☐ s.s \$f1,label

☐ s.s \$f1,label(\$t2)

☐ s.s \$f1,label+100000

☐ s.s \$f1,label+100000(\$t2)

#### *Load & Store addressing modes, pseudo instructions*

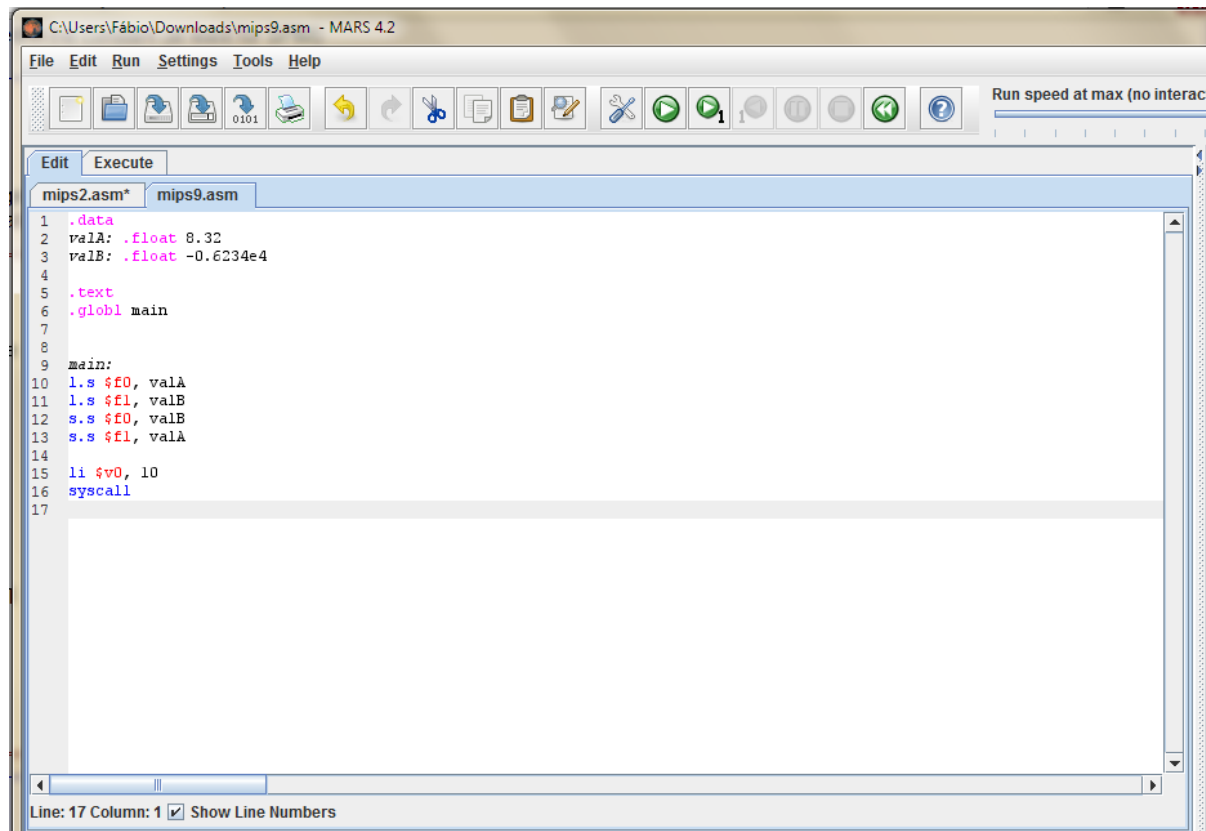
(\$t2)	contents of \$t2
-100	signed 16-bit integer
100	unsigned 16-bit integer
100000	signed 32-bit integer
100 (\$t2)	zero-extended unsigned 16-bit integer added to contents of \$t2
100000 (\$t2)	signed 32-bit integer added to contents of \$t2
label	32-bit address of label
label (\$t2)	32-bit address of label added to contents of \$t2
label+100000	32-bit integer added to label's address
label+100000 (\$t2)	sum of 32-bit integer, label's address, and contents of \$t2



## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### □ Exemplo



The screenshot shows the MARS 4.2 MIPS assembler simulator. The window title is "C:\Users\Fábio\Downloads\mips9.asm - MARS 4.2". The menu bar includes File, Edit, Run, Settings, Tools, and Help. The toolbar contains icons for file operations, execution, and debugging. The status bar at the bottom indicates "Line: 17 Column: 1" and "Show Line Numbers" is checked.

```
1 .data
2 valA: .float 8.32
3 valB: .float -0.6234e4
4
5 .text
6 .globl main
7
8
9 main:
10 l.s $f0, valA
11 l.s $f1, valB
12 s.s $f0, valB
13 s.s $f1, valA
14
15 li $v0, 10
16 syscall
17
```

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

Service	Code in \$v0	Arguments	Returned Value
print integer	1	\$a0 == integer	
print float	2	\$f12 == float	
print double	3	(\$f12, \$f13) == double	
print string	4	\$a0 == address of string	
read integer	5		\$v0 <-- integer
read float	6		\$f0 <-- float
read double	7		(\$f0, \$f1) <-- double
read string	8	\$a0 == buffer address \$a1 == buffer length	
allocate memory	9	\$a0 == number of bytes	\$v0 <-- address
exit	10		

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### □ Instruções Single Precision (Aritméticas)

`abs.s $fd, $fs`  $\Rightarrow$   $\$fd = |\$fs|$

`add.s $fd, $fs, $ft`  $\Rightarrow$   $\$fd = \$fs + \$ft$

`sub.s $fd, $fs, $ft`  $\Rightarrow$   $\$fd = \$fs - \$ft$

`mul.s $fd, $fs, $ft`  $\Rightarrow$   $\$fd = \$fs * \$ft$

`div.s $fd, $fs, $ft`  $\Rightarrow$   $\$fd = \$fs / \$ft$

`neg.s $fd, $fs`  $\Rightarrow$   $\$fd = -\$fs$

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### ❑ Instruções Single Precision (Movimentação de dados)

`mov.s $fd, $fs`

`mtc1 $rs, $fd`

`mfc1 $rd, $fs`

## Instruções: A Linguagem de Máquina

### ► Ponto flutuante em MIPS

#### ❑ Exemplo

The screenshot shows the MARS 4.2 MIPS simulator interface. The main window displays assembly code for a MIPS program that moves data between the CPU and coprocessor 1. The code is as follows:

```
1  ## Move data between the coprocessor and the CPU
2
3
4
5
6  .data
7  val : .float 1.0
8
9  .text
10 .globl main
11
12 main:
13  li    $t0,1          # $t0 <-- 1
14  mfc1  $t0,$f0        # (move to the coprocessor)
15                      # $f0 <-- $t0
16
17  l.s   $f1,val         # $f1 <-- 1.0
18                      # (move from the coprocessor)
19  mfc1  $t1,$f1        # $t1 <-- $f1
20
21  li    $v0,10         # exit
22  syscall
```

On the right side, the Register window shows the state of the MIPS registers. The registers are organized into three columns: Registers, Coproc 1, and Coproc 0. The registers are numbered from \$f0 to \$f31. The values are as follows:

Registers	Coproc 1	Coproc 0
Name	Float	Double
\$f0	1.4E-45	0.007812500000000002
\$f1	1.0	
\$f2	0.0	0.0
\$f3	0.0	
\$f4	0.0	0.0
\$f5	0.0	
\$f6	0.0	0.0
\$f7	0.0	
\$f8	0.0	0.0
\$f9	0.0	
\$f10	0.0	0.0
\$f11	0.0	
\$f12	0.0	0.0
\$f13	0.0	
\$f14	0.0	0.0
\$f15	0.0	
\$f16	0.0	0.0
\$f17	0.0	
\$f18	0.0	0.0
\$f19	0.0	
\$f20	0.0	0.0
\$f21	0.0	
\$f22	0.0	0.0

## Instruções: A Linguagem de Máquina

### ► Instruções em Ponto Flutuante

#### Movimentação de dados

Instrução	Descrição
<b>l.&lt;type&gt; FRdest, mem</b>	Carrega o valor da memória no registro de destino
<b>s.&lt;type&gt; FRsrc, mem</b>	Armazena o conteúdo do registrador na memória
<b>mov.&lt;type&gt; Frdest, FRsrc</b>	Copia o conteúdo do registrador fonte no registrador destino

## Instruções: A Linguagem de Máquina

### ► Instruções em Ponto Flutuante

#### Diretivas

**A: .float     3.14**

**B: .float     0.0**

**C: .doub     6.28**

**D: .double   0.0**

## Instruções: A Linguagem de Máquina

### ► Instruções em Ponto Flutuante

#### Exemplo

**l.s     \$f6, A**

**s.s     \$f6, B**

**l.d     \$f6, C**

**mov.d \$f8, \$f6**

**s.d     \$f8, D**



## Instruções: A Linguagem de Máquina

### ► Instruções em Ponto Flutuante

#### Movimentação de Dados (entre inteiros e ponto flutuante)

<b>mfcI Rdest, FRsrc</b>	Copia o conteúdo do coprocessador I (FPU) – FRsrc no registrador destino
<b>mfcI.d Rdest, FRsrc</b>	Copia os conteúdos do coprocessador I (FPU) – FRsrc e FRsrc+1 nos registradores Rdest e Rdest+1
<b>mtcI FRsrc, Rdest</b>	Copia o conteúdo de Rdest para o registrador do coprocessador I (FPU) - FRsrc
<b>mtcI.d FRsrc, Rdest</b>	Copia o conteúdo dos registradores Rdest e Rdest+1 para os registradores do coprocessador I (FPU) – FRsrc e FRsrc+1

## Instruções: A Linguagem de Máquina

### ► Instruções em Ponto Flutuante

**Exemplos:**

**mtcl \$s0,\$f12**

**mfc1 \$t1,\$f12**

## Instruções: A Linguagem de Máquina

### ► Instruções em Ponto Flutuante

#### Instruções para Conversão Inteiro/Ponto Flutuante

Instrução	Descrição
<b>cvt.d.s FRdest, FRsrc</b>	Converte o valor (ponto flutuante) no registrador FRsrc em um valor ponto flutuante de precisão dupla e carrega no registrador FRdest.
<b>cvt.d.w FRdest, FRsrc</b>	Converte o valor (inteiro) no registrador FRsrc em um valor de ponto flutuante precisão dupla e carrega no registrador FRdest.
<b>cvt.s.d FRdest, FRsrc</b>	Converte um valor ponto flutuante de precisão dupla no registrador FRsrc em um valor ponto flutuante e carrega no registrador FRdest.
<b>cvt.s.w FRdest, FRsrc</b>	Converte um valor inteiro no registrador FRsrc em um valor ponto flutuante e carrega no registrador FRdest.
<b>cvt.w.d FRdest, FRsrc</b>	Converte um valor de precisão dupla no registrador FRsrc em um valor inteiro e carrega no registrador FRdest.
<b>cvt.w.s FRdest, FRsrc</b>	Converte um valor ponto flutuante no registrador FRsrc em um valor inteiro e carrega no registrador FRdest.

## Instruções: A Linguagem de Máquina

### ► Instruções para conversão

#### Exemplo I:

iNum: .word 42

fNum: .float 0.0

lw \$t0, iNum

mtc1 \$t0, \$f6

cvt.s.w \$f8, \$f6

s.s \$f8, fNum

## Instruções: A Linguagem de Máquina

### ► Instruções para conversão

#### Exemplo2:

pi: .double 3.14

intPi: .word 0

l.d \$f10, pi

cvt.w.d \$f12, \$f10

mfc1 \$t1, \$f12

sw \$t1, intPi

## Instruções: A Linguagem de Máquina

### ► Operações Aritméticas em Ponto Flutuante

Instrução	Descrição
<b>add&lt;type&gt; FRdest, FRsrc, FRsrc</b>	$\text{FRdest} = \text{FRsrc} + \text{FRsrc}$
<b>sub&lt;type&gt; FRdest, FRsrc, FRsrc</b>	$\text{FRdest} = \text{FRsrc} - \text{FRsrc}$
<b>mul&lt;type&gt; FRdest, FRsrc, FRsrc</b>	$\text{FRdest} = \text{FRsrc} * \text{FRsrc}$
<b>div&lt;type&gt; FRdest, FRsrc, FRsrc</b>	$\text{FRdest} = \text{FRsrc} / \text{FRsrc}$
<b>rem&lt;type&gt; FRdest, FRsrc, FRsrc</b>	$\text{FRdest} = \text{FRsrc} \% \text{FRsrc}$

## Instruções: A Linguagem de Máquina

### ► Operações Aritméticas em Ponto Flutuante

#### Exemplo

fnum1: .float 6.28318

fnum2: .float 3.14159

fans1: .float 0.0

fans2: .float 0.0

dnum1: .double 42.3

dnum2: .double 73.6

dans1: .double 0.0

dans2: .double 0.0

`l.s $f4, fnum1`

`l.s $f6, fnum2`

`add.s $f8, $f4, $f6`

`s.s $f8, fans1`

## Instruções: A Linguagem de Máquina

### ► Operações Aritméticas em Ponto Flutuante

#### Exemplo

fnum1: .float 6.28318

fnum2: .float 3.14159

fans1: .float 0.0

fans2: .float 0.0

dnum1: .double 42.3

dnum2: .double 73.6

dans1: .double 0.0

dans2: .double 0.0

mul.s \$f10, \$f4, \$f6

s.s \$f10, fans2

l.d \$f4, fnum1

l.d \$f6, fnum2

sub.d \$f8, \$f4, \$f6

s.d \$f10, dans1



## Instruções: A Linguagem de Máquina

### ► Operações Aritméticas em Ponto Flutuante

#### Exemplo

fnum1: .float 6.28318

fnum2: .float 3.14159

fans1: .float 0.0

fans2: .float 0.0

dnum1: .double 42.3

dnum2: .double 73.6

dans1: .double 0.0

dans2: .double 0.0

div.d \$f10, \$f4, \$f6

s.d \$f10, dans2