

Arquitetura de Computadores

ACH2055

Aula 03 – Assembly MIPS (Parte I)

Norton Trevisan Roman
(norton@usp.br)

23 de agosto de 2019

A Arquitetura MIPS

Definição

- *Microprocessor Without Interlocked Pipeline Stages*
 - Desenvolvida pela MIPS Technologies (1984)

A Arquitetura MIPS

Definição

- *Microprocessor Without Interlocked Pipeline Stages*
 - Desenvolvida pela MIPS Technologies (1984)
- Trata-se de uma arquitetura de conjunto de instruções
 - *Instruction Set Architecture – ISA*
 - No caso do MIPS, RISC (veremos mais adiante)

A Arquitetura MIPS

Definição

- *Microprocessor Without Interlocked Pipeline Stages*
 - Desenvolvida pela MIPS Technologies (1984)
- Trata-se de uma arquitetura de conjunto de instruções
 - *Instruction Set Architecture – ISA*
 - No caso do MIPS, RISC (veremos mais adiante)
- Presentes em diversos sistemas
 - Videogames, roteadores etc

A Arquitetura MIPS

Vantagens

- Instruções têm tamanho fixo
 - 32 e 64 bits
 - X86 não tem tamanho fixo
 - São mais fáceis de se manipular, especialmente por uma pipeline (veremos mais adiante)

A Arquitetura MIPS

Vantagens

- Instruções têm tamanho fixo
 - 32 e 64 bits
 - X86 não tem tamanho fixo
 - São mais fáceis de se manipular, especialmente por uma pipeline (veremos mais adiante)
- As instruções seguem um formato geral fixo
 - OPERAÇÃO <lista de operandos>
 - Ex: ADD \$s1, \$s2, \$s3

Programando em MIPS

Simuladores

- Wemips (online)
- <https://rivoire.cs.sonoma.edu/cs351/wemips/>

```
1 # Not sure what to do now? Enter your mips code here
2 # and hit Step (to run one line at a time) or
3 # hit run (to run them all at once)
4
5 # Keep an eye on the register and stack tracker
6 # to see what changes are being made to them.
7
8 # If you want to preload the stack or some registers
9 # with data, you can click on them to change edit them.
10
11 # If you would like more information, check out the user
12 # guide linked in the menu.
13
14
15 ADDI $s0, $zero, 10
16 ADDI $s1, $zero, 9
17 SW $s0, -8($sp)
18 SW $s1, -4($sp)
19 LW $s2, -8($sp)
20 ADDI $s0, $s0, -1
21 ADDI $s0, $s0, -2
22 ADDI $s0, $s0, -3
23 ADDI $s0, $s0, -5
24 ADDI $s0, $s0, -9
25 ADDI $s0, $s0, -14
26 ADDI $s0, $s0, -23
27 ADDI $s0, $s0, -37
28 LB $s3, 0($s0)
```

Step	Run	Enable auto switching			
S	T	A	V	Stack	Log
				s0:	932
				s1:	618
				s2:	720
				s3:	698
				s4:	845
				s5:	773
				s6:	432
				s7:	900

Programando em MIPS

Registradores

- MIPS possui 32 registradores de uso geral
 - Numerados de 0 a 31
 - Em geral referenciados pelo seu número (\$0 a \$31) ou nome (ex: \$t1)

Programando em MIPS

Registradores

- MIPS possui 32 registradores de uso geral
 - Numerados de 0 a 31
 - Em geral referenciados pelo seu número (\$0 a \$31) ou nome (ex: \$t1)
- Embora possam ser usados livremente, há uma convenção em seu uso
 - Seguida por programadores e compiladores
 - Seguir essa convenção permite que um código possa funcionar com outros em MIPS

Programando em MIPS

Registradores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registradores de resultado
a0 – a3	4 – 7	Registradores de argumentos
t0 – t9	8 – 15, 24 – 25	Registradores temporários
s0 – s7	16 – 23	Registradores salvos
k0 – k1	26 – 27	Registradores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

Programando em MIPS

Registadores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registadores de resultado
a0 – a3	4 – 7	Registadores de argumentos
t0 – t9	8 – 15, 24 – 25	Registadores temporários
s0 – s7	16 – 23	Registadores salvos
k0 – k1	26 – 27	Registadores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

zero sempre contém o valor 0 (*hard-wired*)

Programando em MIPS

Registradores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registradores de resultado
a0 – a3	4 – 7	Registradores de argumentos
t0 – t9	8 – 15, 24 – 25	Registradores temporários
s0 – s7	16 – 23	Registradores salvos
k0 – k1	26 – 27	Registradores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

Iniciados em v guardam a avaliação de uma expressão ou os resultados de uma função

Programando em MIPS

Registadores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registadores de resultado
a0 – a3	4 – 7	Registadores de argumentos
t0 – t9	8 – 15, 24 – 25	Registadores temporários
s0 – s7	16 – 23	Registadores salvos
k0 – k1	26 – 27	Registadores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

Iniciados em a guardam argumentos para as rotinas

Programando em MIPS

Registradores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registradores de resultado
a0 – a3	4 – 7	Registradores de argumentos
t0 – t9	8 – 15, 24 – 25	Registradores temporários
s0 – s7	16 – 23	Registradores salvos
k0 – k1	26 – 27	Registradores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

Iniciados em t guardam valores que não precisam ser preservados entre chamadas

Programando em MIPS

Registradores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registradores de resultado
a0 – a3	4 – 7	Registradores de argumentos
t0 – t9	8 – 15, 24 – 25	Registradores temporários
s0 – s7	16 – 23	Registradores salvos
k0 – k1	26 – 27	Registradores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

Iniciados em s
guardam valores
que precisam
ser preservados
entre chamadas

Programando em MIPS

Registadores – Convenção

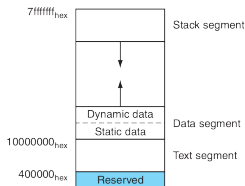
<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registadores de resultado
a0 – a3	4 – 7	Registadores de argumentos
t0 – t9	8 – 15, 24 – 25	Registadores temporários
s0 – s7	16 – 23	Registadores salvos
k0 – k1	26 – 27	Registadores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

Iniciados em **k** são reservados para o kernel do S.O.

Programando em MIPS

Registadores – Convenção

Nome	Número	Uso
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registadores de resultado
a0 – a3	4 – 7	Registadores de argumentos
t0 – t9	8 – 15, 24 – 25	Registadores temporários
s0 – s7	16 – 23	Registadores salvos
k0 – k1	26 – 27	Registadores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address



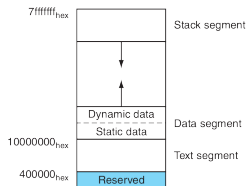
Fonte: [1]

gp aponta para o endereço do fim dos primeiros 32K no segmento de dados estáticos

Programando em MIPS

Registadores – Convenção

Nome	Número	Uso
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registadores de resultado
a0 – a3	4 – 7	Registadores de argumentos
t0 – t9	8 – 15, 24 – 25	Registadores temporários
s0 – s7	16 – 23	Registadores salvos
k0 – k1	26 – 27	Registadores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address



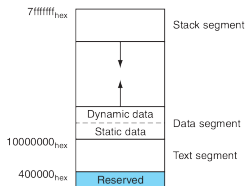
Fonte: [1]

Permitindo assim
acesso rápido aos
primeiros 64K
desse segmento

Programando em MIPS

Registadores – Convenção

Nome	Número	Uso
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registadores de resultado
a0 – a3	4 – 7	Registadores de argumentos
t0 – t9	8 – 15, 24 – 25	Registadores temporários
s0 – s7	16 – 23	Registadores salvos
k0 – k1	26 – 27	Registadores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address



Fonte: [1]

sp guarda o endereço do último local da pilha

Programando em MIPS

Registradores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registradores de resultado
a0 – a3	4 – 7	Registradores de argumentos
t0 – t9	8 – 15, 24 – 25	Registradores temporários
s0 – s7	16 – 23	Registradores salvos
k0 – k1	26 – 27	Registradores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

fp guarda o endereço da primeira palavra da moldura de pilha do procedimento

Programando em MIPS

Registradores – Convenção

<i>Nome</i>	<i>Número</i>	<i>Uso</i>
zero	0	Constante zero
at	1	Reservado para o montador
v0 – v1	2 – 3	Registradores de resultado
a0 – a3	4 – 7	Registradores de argumentos
t0 – t9	8 – 15, 24 – 25	Registradores temporários
s0 – s7	16 – 23	Registradores salvos
k0 – k1	26 – 27	Registradores do Kernel
gp	28	Global Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

ra guarda o endereço de retorno de uma chamada de procedimento

Programando em MIPS

Instruções Aritméticas

- Cada instrução executa uma única operação, possuindo 3 campos
 - 2 operandos e um local para armazenamento do resultado
 - Cada um correspondendo a um dos 32 registradores disponíveis

Programando em MIPS

Instruções Aritméticas

- Cada instrução executa uma única operação, possuindo 3 campos
 - 2 operandos e um local para armazenamento do resultado
 - Cada um correspondendo a um dos 32 registradores disponíveis
- Formato:
 - operação \$resultado, \$operando₁, \$operando₂
 - Ex: add \$t0, \$s2, \$s3

Programando em MIPS

Principais Instruções Aritméticas

- Adição
 - `add $t0, $s2, $s3`
- Subtração
 - `sub $t0, $s2, $s3`
- Multiplicação
 - `mult $t0, $s2, $s3`
- Divisão
 - `div $t0, $s2, $s3`

Programando em MIPS

Principais Instruções Aritméticas

- Adição
 - `add $t0, $s2, $s3`
- Subtração
 - `sub $t0, $s2, $s3`
- Todas com versões para inteiros sem sinal
 - `addu, subu, multu, divu`
- Multiplicação
 - `mult $t0, $s2, $s3`
- Divisão
 - `div $t0, $s2, $s3`

Programando em MIPS

Principais Instruções Aritméticas

- Adição
 - `add $t0, $s2, $s3`
- Subtração
 - `sub $t0, $s2, $s3`
- Todas com versões para inteiros sem sinal
 - `addu, subu, multu, divu`
- Para ponto flutuante etc
 - `add.d, sub.d, mult.d, div.d`
- Multiplicação
 - `mult $t0, $s2, $s3`
- Divisão
 - `div $t0, $s2, $s3`

Programando em MIPS

Constantes (valores imediatos)

- Essas instruções, contudo, trabalham com registradores pré-carregados
 - Como carregar esses registradores?

Programando em MIPS

Constantes (valores imediatos)

- Essas instruções, contudo, trabalham com registradores pré-carregados
 - Como carregar esses registradores?
- Com operadores imediatos
 - `li $s2, 10` (*load immediate* – a constante 10 é armazenada no registrador \$s2)

Programando em MIPS

Constantes (valores imediatos)

- Essas instruções, contudo, trabalham com registradores pré-carregados
 - Como carregar esses registradores?
- Com operadores imediatos
 - `li $s2, 10` (*load immediate* – a constante 10 é armazenada no registrador \$s2)
- Instruções aritméticas também têm versões assim:
 - Ex: `addi $t0, $s2, 4`
 - Presentes apenas para soma e subtração

Programando em MIPS

Constantes (valores imediatos)

- De fato, a instrução `li` não existe
 - Trata-se de uma pseudo-instrução

Programando em MIPS

Constantes (valores imediatos)

- De fato, a instrução `li` não existe
 - Trata-se de uma pseudo-instrução
- Em geral traduzida pelo montador como
 - `addiu destino, 0, valor`
 - `addiu – add immediate unsigned`
 - Ex: `li $s4, 10` \Rightarrow `addiu $s4, $zero, 10`

Programando em MIPS

Estrutura de um Programa MARS

- Dividido em 2 partes: declaração e código

Programando em MIPS

Estrutura de um Programa MARS

- Dividido em 2 partes: declaração e código
- Declaração
 - Usa a diretiva `.data`
 - Itens subsequentes serão armazenados no segmento de dados, no próximo endereço disponível
 - Se for fornecido um endereço (`.data end` → opcional), serão armazenados a partir desse endereço

Programando em MIPS

Estrutura de um Programa MARS

- Dividido em 2 partes: declaração e código
- Declaração
 - Usa a diretiva `.data`
 - Itens subsequentes serão armazenados no segmento de dados, no próximo endereço disponível
 - Se for fornecido um endereço (`.data end` → opcional), serão armazenados a partir desse endereço
 - Formato dos itens:
 - nome: `.tipo_de_armazenagem valor`
 - Ex: `var1: .word 10`

Programando em MIPS

Estrutura de um Programa MARS

- Dividido em 2 partes: declaração e código
 - Declaração
 - Usa a diretiva `.data`
 - Itens subsequentes serão armazenados no segmento de dados, no próximo endereço disponível
 - Se for fornecido um endereço (`.data end` → opcional), serão armazenados a partir desse endereço
 - Formato dos itens:
 - **nome:** `.tipo_de_armazenagem valor`
 - Ex: `var1: .word 10`
- Palavras seguidas por ':' (ex: `var1`) são rótulos, marcando o endereço de memória a seguir (onde 10 está na RAM)

Programando em MIPS

Estrutura de um Programa

- Código
 - Usa a diretiva `.text`
 - Itens subsequentes serão armazenados no segmento de texto
 - Se for fornecido um endereço (`.text end` → opcional), serão armazenados a partir desse endereço
 - Parte do programa em que as instruções a serem executadas são escritas

Programando em MIPS

Estrutura de um Programa

- Código
 - Usa a diretiva `.text`
 - Itens subsequentes serão armazenados no segmento de texto
 - Se for fornecido um endereço (`.text end` → opcional), serão armazenados a partir desse endereço
 - Parte do programa em que as instruções a serem executadas são escritas
- De início, focaremos apenas nessa parte

Programando em MIPS

Um Primeiro Programa

The screenshot shows the MARS 4.5 MIPS assembler simulator. The main window displays the assembly code for a file named `ex1.asm`. The code consists of four lines:

```
1 .text
2 li $s2, 10      #carrega o valor IMEDIATO 10 no registrador s2 (li = load immediate)
3 li $s3, 15      #carrega o valor IMEDIATO 15 no registrador s3
4 add $s1, $s2, $s3 #executa a soma de s2 com s3 e armazena o resultado em s1
5
```

The status bar at the bottom of the editor indicates "Line: 5 Column: 1" and "Show Line Numbers" is checked. Below the editor, the "Mars Messages" window shows the output: "Assemble: assembling /home/norton/academico/disciplinas/usp/Graduacao/Arquitetura/aulas/aula... Assemble: operation completed successfully." The "Registers" window on the right shows the state of the MIPS registers, with all values set to 0x00000000.

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$op	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Programando em MIPS

Um Primeiro Programa

The screenshot displays the MARS 4.5 MIPS assembler simulator. The main window shows the assembly code for a file named `ex1.asm`. The code consists of five lines:

```
1 .text
2 li $s2, 10 #carrega o valor IMEDIATO 10 no registrador s2 (li = load immediate)
3 li $s3, 15 #carrega o valor IMEDIATO 15 no registrador s3
4 add $s1, $s2, $s3 #executa a soma de s2 com s3 e armazena o resultado em s1
5 |
```

A red box highlights the `Edit` and `Execute` buttons in the top-left corner of the editor. A red arrow points from this box to a smaller, magnified view of the editor window, which shows the same code with syntax highlighting: `.text` is pink, `li` is blue, `add` is red, and the register names and values are green.

On the right side of the simulator, the `Registers` window is open, displaying a table of registers and their current values:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

At the bottom of the simulator, the `Mars Messages` window shows the following message:

```
Assemble: assembling /home/norton/academico/disciplinas/usp/Graduacao/Arquitetura/aulas/aula...
Assemble: operation completed successfully.
```


Programando em MIPS

Um Primeiro Programa

The screenshot shows the MARS 4.5 IDE with the following assembly code in the editor:

```
1 .text
2 li $s2, 10      #carrega o valor IMEDIATO 10 no registrador s2 (li = load immediate)
3 li $s3, 15      #carrega o valor IMEDIATO 15 no registrador s3
4 add $s1, $s2, $s3 #executa a soma de s2 com s3 e armazena o resultado em s1
5
```

Annotations in the image:

- A red box highlights the comment on line 4: `#executa a soma`.
- A red arrow points from this box to a larger red box containing the text: `#carrega o valor`, `#carrega o valor`, and `#executa a soma`.
- Below the editor, red text reads: `Comentários iniciam com # e vão até o final da linha`.

The Register Window on the right shows the state of registers:

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

The status bar at the bottom indicates: `Line: 5 Column: 1 Show Line Numbers`. The Mars Messages window shows: `Assemble: assembling /home/norton/academico/disciplinas/usp/Graduacao/Arquitetura/aulas/aula...` and `Assemble: operation completed successfully.`

Programando em MIPS

Um Primeiro Programa

Montamos o programa...

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Mars Messages Run I/O

Clear

Assemble: assembling /home/norton/academico/disciplinas/usp/Graduacao/Arquitetura/aulas/aula03/ex1.asm
Assemble: operation completed successfully.

Programando em MIPS

Um Primeiro Programa

Registers

Name	Number	Coproc 1	Coproc 0	Value
\$zero	0			0x00000000
\$at	1			0x00000000
\$v0	2			0x00000000
\$v1	3			0x00000000
\$a0	4			0x00000000
\$a1	5			0x00000000
\$a2	6			0x00000000
\$a3	7			0x00000000
\$t0	8			0x00000000
\$t1	9			0x00000000
\$t2	10			0x00000000
\$t3	11			0x00000000
\$t4	12			0x00000000
\$t5	13			0x00000000
\$t6	14			0x00000000
\$t7	15			0x00000000
\$s0	16			0x00000000
\$s1	17			0x00000000
\$s2	18			0x00000000
\$s3	19			0x00000000
\$s4	20			0x00000000
\$s5	21			0x00000000
\$s6	22			0x00000000
\$s7	23			0x00000000
\$t8	24			0x00000000
\$t9	25			0x00000000
\$k0	26			0x00000000
\$k1	27			0x00000000
\$gp	28			0x10000000
\$sp	29			0x7fffffc
\$fp	30			0x00000000
\$ra	31			0x00000000
pc				0x00400000
hi				0x00000000
lo				0x00000000

Mars Messages

Run I/O

Assemble: operation completed successfully.

Programando em MIPS

Um Primeiro Programa

E então rodamos

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$fp	28	0x10000000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages Run I/O

Assemble: operation completed successfully.

Programando em MIPS

Um Primeiro Programa

Podemos visualizar o conteúdo dos registradores →

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000019
\$s2	18	0x00000000
\$s3	19	0x0000000f
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$fp	28	0x10000000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x0040000c
hi		0x00000000
lo		0x00000000

Programando em MIPS

Instruções Relacionais

- Possuem o mesmo formato das aritméticas
 - operação \$resultado, \$operando₁, \$operando₂
 - Ex: `slt $t0, $s2, $s3`

Programando em MIPS

Instruções Relacionais

- Possuem o mesmo formato das aritméticas
 - operação \$resultado, \$operando₁, \$operando₂
 - Ex: slt \$t0, \$s2, \$s3
- Existem, contudo, apenas 2 instruções
 - SLT e SLTI
 - De fato, há mais duas, relacionadas a desvios, que veremos mais adiante

Programando em MIPS

Instruções Relacionais

- SLT – *Set on Less Than*
 - `slt $t0, $s1, $s2`
 - \$t0 será 1 se $\$s1 < \$s2$. Do contrário, \$t0 será 0
 - Equivale a $\$st0 \leftarrow \$s1 < \$s2$

Programando em MIPS

Instruções Relacionais

- *SLT – Set on Less Than*
 - `slt $t0, $s1, $s2`
 - `$t0` será 1 se `$s1 < $s2`. Do contrário, `$t0` será 0
 - Equivale a `$st0 ← $s1 < $s2`
- *SLTI – Set on Less Than Immediate (com sinal)*
 - `slti $t0, $s1, 10`
 - Mesmo comportamento de `slt`
 - Existe também a versão para *unsigned int*: `sltiu`

Programando em MIPS

Instruções Lógicas

- São 7 as instruções lógicas em MIPS
 - Deslocamento à esquerda (SLL)
 - Deslocamento à direita (SRL)
 - *And* bit a bit (AND, ANDI)
 - *Or* bit a bit (OR, ORI)
 - *Nor* bit a bit (NOR)

Programando em MIPS

Instruções Lógicas

- SLL – *Shift Left Logical*
 - sll resultado, origem, deslocamento
 - sll \$t2, \$s0, 4
 - Desloque o conteúdo de \$s0 4 bits para a esquerda e armazene o resultado em \$t2

Programando em MIPS

Instruções Lógicas

- *SLL – Shift Left Logical*
 - sll resultado, origem, deslocamento
 - sll \$t2, \$s0, 4
 - Desloque o conteúdo de \$s0 4 bits para a esquerda e armazene o resultado em \$t2
- *SRL – Shift Right Logical*
 - srl resultado, origem, deslocamento
 - srl \$t2, \$s0, 2
 - Desloque o conteúdo de \$s0 2 bits para a direita e armazene o resultado em \$t2

Programando em MIPS

Instruções Lógicas

- SLL – *Shift Left Logical*

- sll resultado, origem, deslocamento

- sll \$t2, \$s0, 4

- Desloque o conteúdo de \$s0 4 bits para a esquerda e armazene o resultado em \$t2

São chamadas lógicas por não preservarem sinal nem distinguirem expoente de mantissa

- SRL – *Shift Right Logical*

- srl resultado, origem, deslocamento

- srl \$t2, \$s0, 2

- Desloque o conteúdo de \$s0 2 bits para a direita e armazene o resultado em \$t2

Programando em MIPS

Instruções Lógicas

- SLL – *Shift Left Logical*

- sll resultado, origem, deslocamento

- sll \$t2, \$s0, 4

- Desloque o conteúdo de \$s0 4 bits para a esquerda e armazene o resultado em \$t2

Os bits são simplesmente movidos sem preocupação com seu significado

- SRL – *Shift Right Logical*

- srl resultado, origem, deslocamento

- srl \$t2, \$s0, 2

- Desloque o conteúdo de \$s0 2 bits para a direita e armazene o resultado em \$t2

Programando em MIPS

Instruções Lógicas

- AND
 - `and resultado, origem1, origem2`
 - `and $t0, $s1, $s2`
 - \$t0 recebe o resultado de um *AND* bit a bit entre \$s1 e \$s2

Programando em MIPS

Instruções Lógicas

- AND
 - and resultado, origem₁, origem₂
 - and \$t0, \$s1, \$s2
 - \$t0 recebe o resultado de um *AND* bit a bit entre \$s1 e \$s2
- ANDI – *AND Immediate*
 - andi resultado, origem₁, valor
 - andi \$t0, \$s2, 10
 - \$t0 recebe o resultado de um *AND* bit a bit entre \$s2 e 10

Programando em MIPS

Instruções Lógicas

- OR
 - `or resultado, origem1, origem2`
 - `or $t0, $s1, $s2`
 - \$t0 recebe o resultado de um *OR* bit a bit entre \$s1 e \$s2

Programando em MIPS

Instruções Lógicas

- OR
 - or resultado, origem₁, origem₂
 - or \$t0, \$s1, \$s2
 - \$t0 recebe o resultado de um *OR* bit a bit entre \$s1 e \$s2
- ORI – *OR Immediate*
 - ori resultado, origem₁, valor
 - ori \$t0, \$s2, 10
 - \$t0 recebe o resultado de um *OR* bit a bit entre \$s2 e 10

Programando em MIPS

Instruções Lógicas

- NOR
 - Implementado assim para seguir o padrão de 2 operandos
 - Podemos, contudo, simular o NOT usando NOR:

A NOR 0
NOT (A OR 0)
NOT A

Programando em MIPS

Instruções Lógicas

- NOR
 - Implementado assim para seguir o padrão de 2 operandos
 - Podemos, contudo, simular o NOT usando NOR:

```
A NOR 0
NOT (A OR 0)
NOT A
```

- `nor resultado, origem1, origem2`
- `nor $t0, $s1, $zero`
 - \$t0 recebe o resultado de um *NOT* bit a bit em \$s1
 - Lembre-se que todos os 32 (ou 64) bits são invertidos

Programando em MIPS

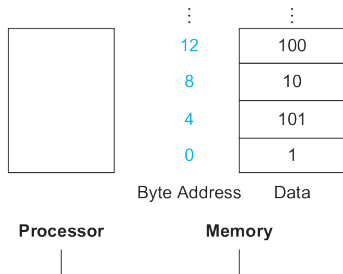
Endereçamento de Memória

- MIPS endereça cada byte de memória
 - Com 32 bits, endereços de palavras adjacentes diferem em 4
 - Temos assim $2^{32} \div 4 = 2^{30}$ palavras de memória

Programando em MIPS

Endereçamento de Memória

- MIPS endereça cada byte de memória
 - Com 32 bits, endereços de palavras adjacentes diferem em 4
 - Temos assim $2^{32} \div 4 = 2^{30}$ palavras de memória
- A memória então é vista como um arranjo unidimensional
 - Com o endereço funcionando como índice

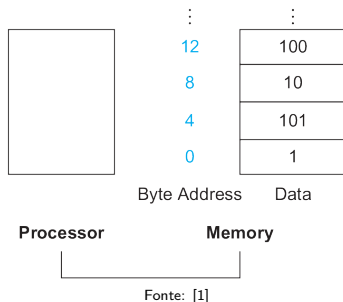


Fonte: [1]

Programando em MIPS

Endereçamento de Memória

- Mas e como os bytes estão arranjados dentro da palavra?
- O mais significativo (*big-endian*) ou o menos significativo (*little-endian*) no endereço mais baixo?



Programando em MIPS

Endereçamento de Memória

- Mas e como os bytes estão arranjados dentro da palavra?
- O mais significativo (*big-endian*) ou o menos significativo (*little-endian*) no endereço mais baixo?

Big-endian

Byte #			
0	1	2	3

Little-endian

Byte #			
3	2	1	0

Fonte: [1]

Programando em MIPS

Endereçamento de Memória

- Mas e como os bytes estão arranjados dentro da palavra?
- O mais significativo (*big-endian*) ou o menos significativo (*little-endian*) no endereço mais baixo?
- MIPS é do tipo *big-endian*

Big-endian

Byte #			
0	1	2	3

Little-endian

Byte #			
3	2	1	0

Fonte: [1]

Programando em MIPS

Instruções de Acesso à Memória

- MIPS possui diversas instruções de acesso à memória
 - As principais sendo `lw` e `sw`

Programando em MIPS

Instruções de Acesso à Memória

- MIPS possui diversas instruções de acesso à memória
 - As principais sendo `lw` e `sw`
- *LW – Load Word*
 - Transfere uma palavra da RAM para um registrador
 - `lw registrador, deslocamento(registrador_base)`
 - `lw $t0, 30($s0)`
 - Carregue em `$t0` a palavra no endereço correspondente a `$s0 + 30`
 - O deslocamento (aqui 30) pode ser negativo

Programando em MIPS

Instruções de Acesso à Memória

- *SW – Store Word*
 - Transfere uma palavra de um registrador para a RAM
 - `sw registrador, deslocamento(registrador_base)`
 - `sw $t0, 30($s0)`
 - Armazene o conteúdo de `$t0` no endereço correspondente a `$s0 + 30`
 - Novamente, o deslocamento pode ser negativo

Programando em MIPS

Instruções de Acesso à Memória

- E como sabemos o endereço de memória?
 - Ou especificamos, carregando o registrador com `li` (cuidado!!!)
 - Ou deixamos que o montador tome conta disso

Programando em MIPS

Instruções de Acesso à Memória

- E como sabemos o endereço de memória?
 - Ou especificamos, carregando o registrador com `li` (cuidado!!!)
 - Ou deixamos que o montador tome conta disso
- Como?
 - Usando as diretivas de montador MIPS
 - Ex:

```
.data  
v1:  .word 20
```

Programando em MIPS

Instruções de Acesso à Memória

The screenshot displays the MARS 4.5 MIPS simulator interface. The main window shows the assembly code for 'ex1.asm' with the following instructions:

```
1 .data
2 v1: .word 20 #armazena 20 no segmento de dados
3
4 .text
5 li $s2, 10 #carrega o valor IMEDIATO 10 no registrador s2 (li = load immediate)
6 li $s3, 15 #carrega o valor IMEDIATO 15 no registrador s3
7 add $s1, $s2, $s3 #executa a soma de s2 com s3 e armazena o resultado em s1
8 sll $t0, $s3, $s2
9 sll $t1, $s2, 5
10 sll $t0, $s2, 3
11 and $t0, $s2, $s3
12 andi $t0, $s2, 6
13 or $t0, $s2, $s3
14 ori $t0, $s2, 6
15 nor $t0, $s2, $zero
16 lw $t1, v1
17
```

The Register Window on the right shows the state of the MIPS registers:

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0xffffffff
\$t1	9	0x00000014
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000019
\$s2	18	0x0000000a
\$s3	19	0x0000000f
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffcfc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400034
hi		0x00000000
lo		0x00000000

The status bar at the bottom indicates: "Mars Messages Run I/O" and "Clear -- program is finished running (dropped off bottom) --".

Programando em MIPS

Instruções de Acesso à Memória

The screenshot shows the MARS 4.5 MIPS simulator interface. The main window displays assembly code for a program named `ex1.asm`. The code includes a data declaration and several instructions. A callout box highlights the first three lines of the code, and another callout box highlights lines 15-17. The register window on the right shows the state of the MIPS registers.

```
1 .data
2 v1: .word 20 #armazena 20 no segmento de dados
3
4 .text
5 li $s2, 10 #carrega o valor 10 no registro $s2
6 li $s3, 15 #carrega o valor 15 no registro $s3
7 add $s1, $s2, $s3 #executa a soma de $s2 e $s3
8 sll $t0, $s2, 5
9 sll $t1, $s2, 5
10 sll $t0, $s2, 3
11 and $t0, $s2, $s3
12 andi $t0, $s2, 6
13 or $t0, $s2, $s3
14 sll $t0, $s2, 6
15 nor $t0, $s2, $zero
16 lw $t1, v1
17 |
```

Registers window:

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0xffffffff
\$t1	9	0x00000014
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000019
\$s2	18	0x0000000a
\$s3	19	0x0000000f
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400034
hi		0x00000000
lo		0x00000000

Programando em MIPS

Instruções de Acesso à Memória

The screenshot shows the MARS 4.5 MIPS simulator interface. The main window displays assembly code for a program named `ex1.asm`. The code includes a data declaration, a text section, and several instructions. A callout box highlights the first three lines of the code, and another callout box highlights lines 15-17, with a text annotation explaining that `lw` is a pseudo-instruction.

```
1 .data
2 v1: .word 20 #armazena 20 no segmento de dados
3
4 .text
5 li $s2, 10 #carrega o valor 10 no registrador $s2
6 li $s3, 15 #carrega o valor 15 no registrador $s3
7 add $s1, $s2, $s3 #executa a soma de $s2 e $s3
8 sll $t0, $s3, 5
9 sll $t1, $s2, 5
10 sll $t0, $s2, 3
11 and $t0, $s2, $s3
12 andi $t0, $s2, 6
13 or $t0, $s2, $s3
14 andi $t0, $s2, 6
15 nor $t0, $s2, $zero
16 lw $t1, v1
17 |
```

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0xffffffff
\$t1	9	0x00000014
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000019
\$s2	18	0x0000000a
\$s3	19	0x0000000f
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400034
hi		0x00000000
lo		0x00000000

Mars Messages Run I/O

Clear -- program is finished running (dropped off bottom) --

Programando em MIPS

Instruções de Acesso à Memória

The screenshot shows the MARS 4.5 MIPS simulator interface. The main window displays assembly code with the following instructions highlighted in red:

```
0x02400027: andi $t1, $s7, $zero
0x02400028: ori $t1, $t1, $zero
0x02400029: nor $t1, $t1, $zero
0x0240002a: lw $t1, v1
```

The registers window on the right shows the state of the MIPS registers:

Name	Number	Value
\$zero	0	0x00000000
\$t1	1	0x10010000
\$t0	2	0x00000000
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$a4	8	0x00000000
\$t1	9	0x00000014
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$t8	16	0x00000000
\$s1	17	0x00000019
\$s2	18	0x0000000a
\$s3	19	0x0000000f
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400034
hi		0x00000000
lo		0x00000000

The Data Segment window shows memory addresses and values:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x10010000	0x00000014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001000c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001001c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

The MARS Messages window shows the message: "-- program is finished running (dropped off bottom) --".

Programando em MIPS

Arranjos

- Suponha que temos um arranjo em memória

```
.data
```

```
arr:  .word 2, 0, -1, 5, -3, 2, 2, 10, 9, -2
```

Programando em MIPS

Arranjos

- Suponha que temos um arranjo em memória

```
.data
```

```
arr: .word 2, 0, -1, 5, -3, 2, 2, 10, 9, -2
```

- Como acessar o elemento `arr[3]`?
 - Precisamos armazenar o endereço-base do arranjo em um registrador, para calcular o deslocamento
 - `la $t1, arr`
 - Pseudo-instrução que carrega em `$t1` o endereço representado por `arr`

Programando em MIPS

Arranjos

- Como acessar o elemento `arr[3]` (cont.)?
 - `lw $t2, 3($t1)?`

Programando em MIPS

Arranjos

- Como acessar o elemento `arr[3]` (cont.)?
 - `lw $t2, 3($t1)?`
 - Mas armazenamos palavras, não bytes
 - O índice precisa ser multiplicado pelo tamanho da palavra

Programando em MIPS

Arranjos

- Como acessar o elemento `arr[3]` (cont.)?
 - `lw $t2, 3($t1)?`
 - Mas armazenamos palavras, não bytes
 - O índice precisa ser multiplicado pelo tamanho da palavra
 - Então `lw $t2, 12($t1)`

```
.data
arr: .word 2, 0, -1, 5, -3, 2, 2, 10, 9, -2
.text
la $t1, arr
lw $t2, 12($t1)
```

Programando em MIPS

Codificação das Instruções

- MIPS possui 3 famílias de instruções:
 - Instruções do Tipo R (*Register*)
 - `add $s1, $s2, $s3`
 - `sll $t2, $s0, 4`

Programando em MIPS

Codificação das Instruções

- MIPS possui 3 famílias de instruções:
 - Instruções do Tipo R (*Register*)
 - `add $s1, $s2, $s3`
 - `sll $t2, $s0, 4`
 - Instruções do Tipo I (*Immediate*)
 - `addi $t0, $s2, 4`
 - `lw $t2, 12($t1)`

Programando em MIPS

Codificação das Instruções

- MIPS possui 3 famílias de instruções:
 - Instruções do Tipo R (*Register*)
 - `add $s1, $s2, $s3`
 - `sll $t2, $s0, 4`
 - Instruções do Tipo I (*Immediate*)
 - `addi $t0, $s2, 4`
 - `lw $t2, 12($t1)`
 - Instruções do Tipo J (*Jump*)
 - Usadas em desvios (mais adiante)

Programando em MIPS

Codificação das Instruções

- Instruções Tipo R (*Register*)

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

Fonte: [1]

Programando em MIPS

Codificação das Instruções

- Instruções Tipo R (*Register*)



Código da operação (*opcode*)

Programando em MIPS

Codificação das Instruções

- Instruções Tipo R (*Register*)



Fonte: [1]

Primeiro registrador-fonte da operação



Programando em MIPS

Codificação das Instruções

- Instruções Tipo R (*Register*)



Segundo registrador-fonte da operação

Programando em MIPS

Codificação das Instruções

- Instruções Tipo R (*Register*)



Fonte: [1]

Registrador-destino (recebe o resultado da operação)

Programando em MIPS

Codificação das Instruções

- Instruções Tipo R (*Register*)



Fonte: [1]

Shift amount – deslocamento (usado em instruções como `sll` e `srl`)

Programando em MIPS


Codificação das Instruções

- Instruções Tipo R (*Register*)



Fonte: [1]

Código de função – seleciona a variante específica da operação no campo *op*



Programando em MIPS

Codificação das Instruções

- Instruções Tipo R (*Register*)

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

Fonte: [1]

Campos não usados são carregados com zero

Programando em MIPS

Codificação das Instruções

- Instruções Tipo I (*Immediate*)



Fonte: [1]

Programando em MIPS

Codificação das Instruções

- Instruções Tipo I (*Immediate*)



Fonte: [1]



Código da operação (*opcode*)

Programando em MIPS

Codificação das Instruções

- Instruções Tipo I (*Immediate*)



Fonte: [1]

Registrador-fonte da operação



Programando em MIPS

Codificação das Instruções

- Instruções Tipo I (*Immediate*)



Fonte: [1]

Registrador de destino da operação

Programando em MIPS

Codificação das Instruções

- Instruções Tipo I (*Immediate*)



Fonte: [1]

Valor (constante ou endereço) usado na operação

Programando em MIPS

Codificação das Instruções

- Instruções Tipo I (*Immediate*)



Fonte: [1]

Note que isso nos dá um limite de $\pm 2^{15}$ bytes de deslocamento (há o bit de sinal) em `lw $reg1, desl($reg2)`

Programando em MIPS

Codificação das Instruções

- Instruções Tipo I (*Immediate*)



Por isso, ao apontar para o byte 32K no segmento de dados, \$gp dá acesso aos primeiros 64K desse segmento

Programando em MIPS

Codificação das Instruções

- Instruções Tipo J (*Jump*)



Fonte: Adaptada de [1]

Programando em MIPS

Codificação das Instruções

- Instruções Tipo J (*Jump*)



Fonte: Adaptada de [1]

Código da operação (*opcode*)

Programando em MIPS

Codificação das Instruções

- Instruções Tipo J (*Jump*)



6 bits

26 bits

Fonte: Adaptada de [1]

Endereço de memória

Programando em MIPS

Codificação das Instruções

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
op	rs	rt	constant or address		
6 bits	5 bits	5 bits	16 bits		
op	address				
6 bits	26 bits				

Fonte: Adaptada de [1]

- Os formatos são diferenciados no circuito pelo valor no campo *op*

Programando em MIPS

Pseudo-instruções

- Instruções adicionadas pelo montador, para simplificar algumas operações
 - Tratadas por ele como se fossem instruções legítimas
 - Traduzidas para instruções reais

Programando em MIPS

Pseudo-instruções

- Instruções adicionadas pelo montador, para simplificar algumas operações
 - Tratadas por ele como se fossem instruções legítimas
 - Traduzidas para instruções reais
- Exemplos:

<i>Pseudo-instrução</i>	<i>Expansão</i>
li \$s4, 10	addiu \$s4, \$zero, 10
move \$t0, \$t1	add \$t0, \$zero, \$t1

Referências

- 1 Patterson, D.A.; Hennessy, J.L. (2013): Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann. 5ª ed.
 - Para detalhes sobre MIPS consulte também o Apêndice A
- 2 <https://www.embarcados.com.br/arquitetura-de-conjunto-de-instrucoes-mips/>
 - Curso bastante completo, em português
- 3 <https://sweetcode.io/building-first-simple-program-mips-assembly-language/>
 - Tutorial básico passo a passo
- 4 <https://sites.cs.ucsb.edu/~franklin/64/lectures/mipsassemblytutorial.pdf>
 - Tutorial mais profundo

Referências

1 WEMIPS

- <https://rivoire.cs.sonoma.edu/cs351/wemips/>
- Simulador MIPS online

2 MARS (*MIPS Assembler and Runtime Simulator*)

- <http://courses.missouristate.edu/kenvollmar/mars/>
- Simulador MIPS (.jar, multiplataforma)

3 https://en.wikipedia.org/wiki/Logical_shift

- Diferença entre deslocamento lógico e aritmético