

# Aula 3

## Organização de Computadores Conjunto de Instruções do MIPS

Profa. Débora Matos

# Arquitetura MIPS

- Arquitetura tipo RISC
- Versões de 32 e 64 bits (veremos apenas da versão de 32 bits)
- MIPS são encontrados em produtos da ATI Technologies, NEC, Nintendo, Cisco, Silicon Graphics, Sony, Texas Instrument, Toshiba, impressoras HP e Fuji, etc.

# Conjunto de Instruções do MIPS

- ISA (Instruction Set Architecture) do MIPS possui 3 formatos de instruções;
- Todas as instruções com 3 operandos;
- Todas as instruções têm tamanho de 32 bits (4 bytes);
- Após uma instrução ser buscada na memória e armazenada no IR, o PC é incrementado em 4 bytes, porque?
  - Para que a CPU tenha o endereço da próxima instrução a ser executada;

# Conjunto de Instruções do MIPS

- MIPS possui 32 registradores:

Exemplos:

\$s0 - \$s7, \$t0 - \$t9, \$zero, \$a0 - \$a3, \$v0 - \$v1,  
\$gp, \$sp, \$ra, \$fp, \$at

Nome	Número do registrador	Uso
\$zero	0	O valor constante 0
\$v0-\$v1	02-03	Valores para resultados e avaliação de expressões
\$a0-\$a3	04-07	Argumentos
\$t0-\$t7	08-15	Temporários
\$s0-\$s7	16-23	Valores salvos
\$t8-\$t9	24-25	Mais temporários
\$gp	28	Ponteiro global
\$sp	29	Ponteiro de pilha
\$fp	30	Ponteiro de quadro
\$ra	31	Endereço de retorno

Registrador 1 (\$at) reservado para o assembler, 26-27 para o sistema operacional

# Montagem de Linguagem (Assembly)

- No MIPS são definidos os registradores em assembly como segue:

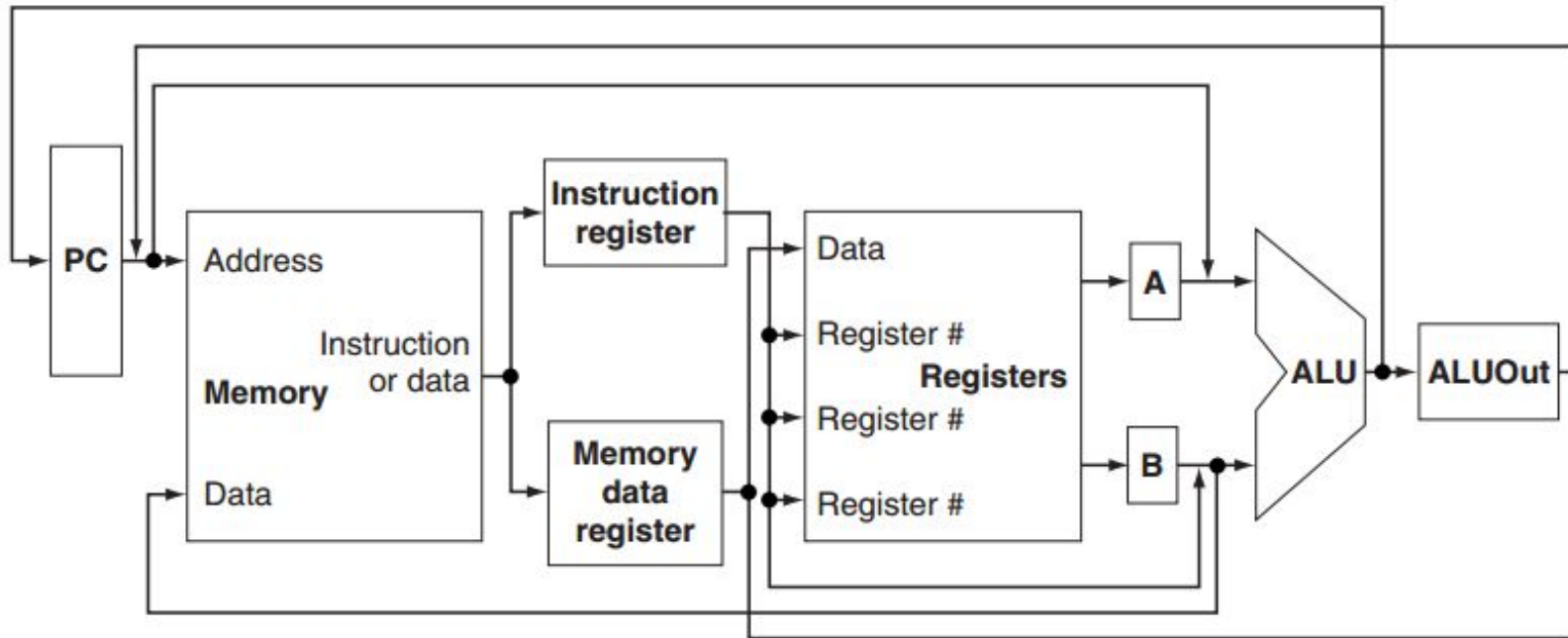
**\$s0, \$s1, \$s2 ....** => registradores para variáveis de programa;

**\$t0, \$t1, \$t2...** => registradores temporários;

É tarefa do compilador associar variáveis do programa aos registradores.

Programa em C	Assembly MIPS
$f = (g + h) - (i + j);$	<b>add \$t0,\$s1,\$s2 add \$t1,\$s3,\$s4 sub \$s0,\$t0,\$t1</b>

# Arquitetura MIPS

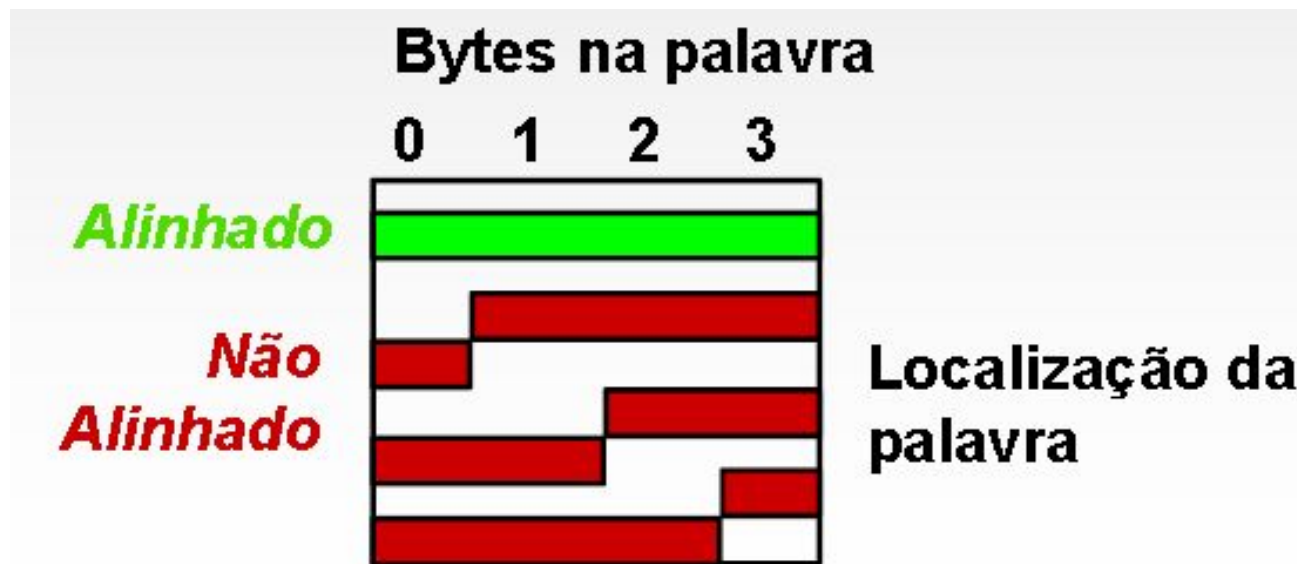


## ➤ Componentes básicos:

- ✓ Contador de programa (PC)
- ✓ Memória
- ✓ Banco de registradores
- ✓ Unidade lógica e aritmética (ALU)
- Unidade de controle
- Registrador de instruções (IR)
- Registrador de dados da memória
- Outros registradores internos

# Instruções MIPS – armazenamento na memória

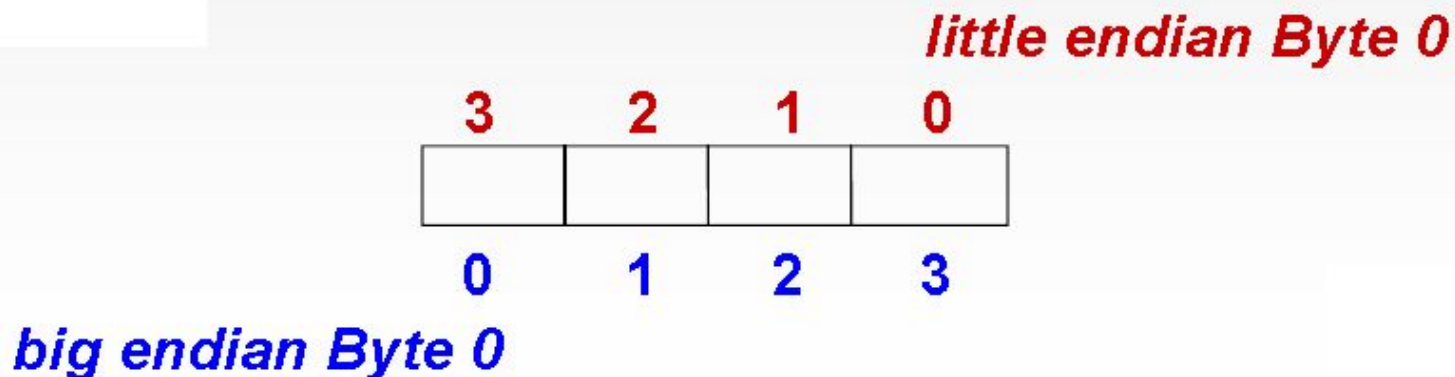
- MIPS exige que todas as palavras comecem em endereços que são múltiplos de 4 bytes;
- Alinhamento: objetos devem estar em um endereço que é um múltiplo do seu tamanho;



# Instruções MIPS – armazenamento na memória

- Dois sistemas para numeração dos Bytes dentro uma palavra:
  - **Big Endian** - Byte mais à esquerda marca endereço da word
  - **Little Endian** – Byte mais à direita marca endereço da word.

**MIPS usa Big Endian.**





# Instruções MIPS - armazenamento na memória

- **Big Endian** - Byte mais à esquerda marca endereço da word
- **Little Endian** – Byte mais à direita marca endereço da word.

Exemplos:

**Big Endian:**

- IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA

**Little Endian:**

- Intel 80x86, DEC Vax, DEC Alpha

# Montagem de Linguagem (Assembly)

- Todas as instruções aritméticas e lógicas com três operandos
- A ordem dos operandos é fixa (destino primeiro);

[label: ] op-code [operando\_dest], [operando], [operando] [#comentario]

- Sintaxe de instruções assembly:
  1. Label: opcional, identifica bloco do programa
  2. Código de operação: indicado por um Mnemônico
  3. Operandos: Registradores ou memória
  4. Comentários: opcional, tudo que vem depois do #

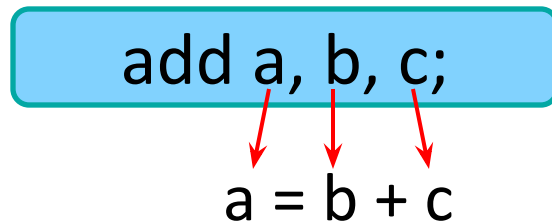
# **Princípios de Projeto da arquitetura do MIPS**

# Arquitetura MIPS

## Princípio de Projeto 1: Simplicidade favorece regularidade.

- Cada instrução aritmética do MIPS realiza apenas 1 operação e sempre precisa ter exatamente 3 variáveis;

add a, b, c;



$a = b + c$

- Essa definição segue o modelo RISC de forma a manter o HW simples;
- Mais que três operandos por instrução exigiria um projeto de hardware mais complicado.



Comente pelo menos 3 situações que precisariam ser consideradas no projeto, se uma instrução aritmética operasse com 4 operandos:

# Montagem de Linguagem (Assembly)

- Exemplo de comando em linguagem C:

```
a = b + c + d + e;
```

- Como no MIPS somente 1 operação é realizada por instrução, são necessárias várias instruções. Exemplo de assembly:

```
add a, b, c  
add a, a, d  
add a, a, e
```

# Montagem de Linguagem (Assembly)

- Exemplo 2 (comando em linguagem C):

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

- O compilador precisa desmembrar essa instrução em várias instruções assembly.  
Correspondência de instruções assembly MIPS:

```
add t0, g, h  
add t1, i, j  
sub f, t0, t1
```

# Montagem de Linguagem (Assembly)

**Princípio de Projeto 2:** Menor significa mais rápido.

Exemplo: uso de um banco de registradores (que é muito menor do que uma memória);

Quais seriam os impactos em fazer-se o uso de um banco de registradores maior ou menor?

- Uma quantidade muito grande de registradores pode aumentar o tempo de ciclo de clock;
- Mais de 32 registradores requereriam mais bits para endereça-los, afetando o número total de bits definido para as instruções (32 bits).



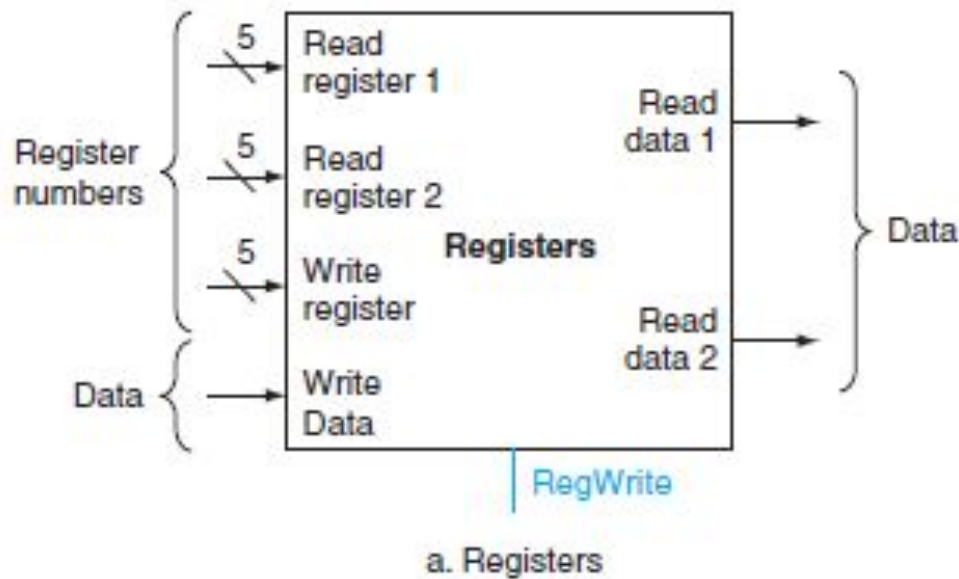
# Banco de Registradores

Dois registradores de leitura:

- Registrador de leitura 1 (Read register 1)
- Registrador de leitura 2 (Read register 2)

Um registrador para escrita:

- Write Register



# Banco de Registradores

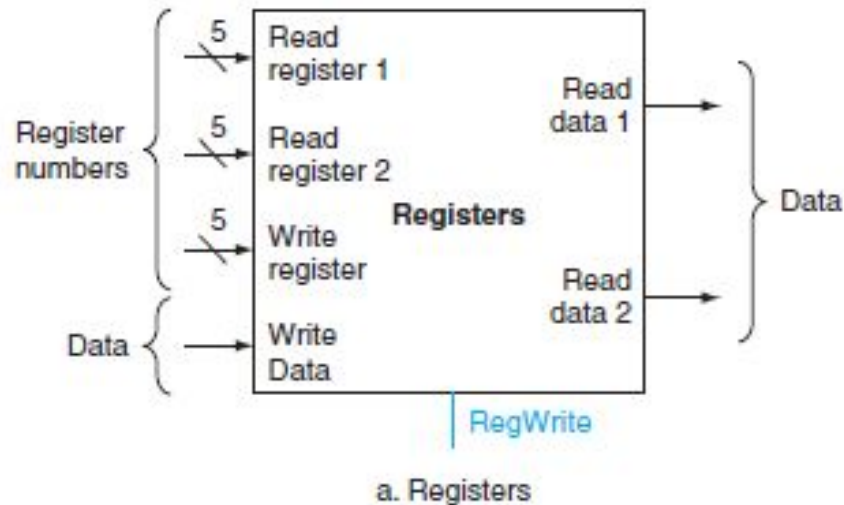
Leitura de duas palavras por vez:

- Read data 1
- Read data 2

Escrita de uma palavra:

- Write data

Sinal de habilita para escrita no banco: *RegWrite*.



# Montagem de Linguagem (Assembly)

- Muitos programas possuem mais variáveis do que registradores.
- Assim, o compilador é responsável por manter as variáveis mais utilizadas nos registradores;
- As demais, são colocadas na memória;

## Vantagens:

Quando os dados estão nos registradores, o MIPS acessa **2 operandos** em registradores, atua sobre eles e escreve o resultado em um **outro registrador muito mais rapidamente do que se tivesse que fazer acessos à memória.**

# Montagem de Linguagem (Assembly)

- Considere que ao invés de os operandos estarem em registradores, como no assembly abaixo, eles estivessem na memória.

```
add $t0, $s0, $s1
```

- Qual seria o assembly?

```
lw $s0, 0($s2)  # carrega o 1° op em $s0
```

```
lw $s1, 0($s3)  # carrega o 2° op em $s1
```

```
add $t0, $s0, $s1  # soma $s0 com $s1 e salva em $t0
```

```
sw $t0, 0($s4)  # armazena o resultado na memória
```

- Para conseguir o melhor desempenho, os compiladores precisam usar os registradores de modo eficaz.

# Montagem de Linguagem (Assembly)

- Em **assembly**, estamos manipulando **registradores** do MIPS;
- Em **código C** (sem compilação), estamos manipulando **posições da memória**;
- A associação entre posições da memória e registradores é realizada pelo compilador C;

# Instruções de movimentação de dados

## Load e Store

- lw : instrução de movimentação de dados da memória para registrador ( load word )
- sw: instrução de movimentação de dados do registrador para a memória ( store word )

Exemplo:

Seja A um array de 100 palavras.

Considere o comando C abaixo:

$$g = h + A[8];$$

Qual seria o corresponde código assembly MIPS?

# Instruções de movimentação de dados

## Formato de Instrução lw – load word

$g = h + A[8];$

O compilador associou à variável:

- g, o registrador \$s1
  - h, o registrador \$s2
  - endereço base do vetor, \$s3
- 2) Transferir A[8] para um registrador
  - 3) O endereço de A[8] é a soma do registrador base com a soma do índice
  - 4) Os dados devem ser colocados em um registrador temporário para uso na próxima instrução.

# Instruções de movimentação de dados

## Formato de Instrução lw – load word

$g = h + A[8];$

- 1) Transferir  $A[8]$  para um registrador
- 2) O endereço de  $A[8]$  é a soma do registrador base com a soma do índice
- 3) Os dados devem ser colocados em um registrador temporário para uso na próxima instrução.

lw \$t0, 8(\$s3)

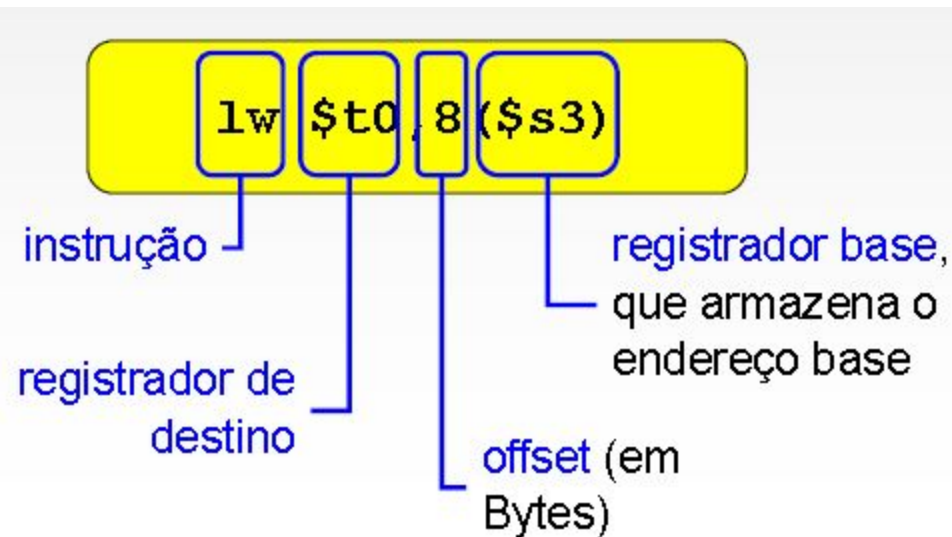
\$t0 recebe  $A[8]$

\$s3 recebe o valor de A



# Instruções de movimentação de dados

Copiar dados de → para	Instrução
Memória → Registrador	load word (lw)
Registrador → Memória	store word (sw)



# Instruções de movimentação de dados

**Código C:**

$\$s1$     $\$s2$     $\$t0$   
↑   ↑   ↑  
 $g = h + A[8];$

**Correspondente assembly MIPS:**

```
lw $t0, 8($s3)   # $t0 recebe A[8]  
add $s1, $s2, $t0   # $s1 recebe a soma de h + A[8]
```

- No MIPS as words precisam começar em endereços que sejam múltiplos de 4 (words são armazenadas de 4 em 4 bytes);
- Sendo assim o offset a ser somado ao registrador base \$s3 precisa ser multiplicado por 4, já que cada palavra possui 4 bytes e a memória está dividida em bytes.

# Instruções de movimentação de dados

Código C:

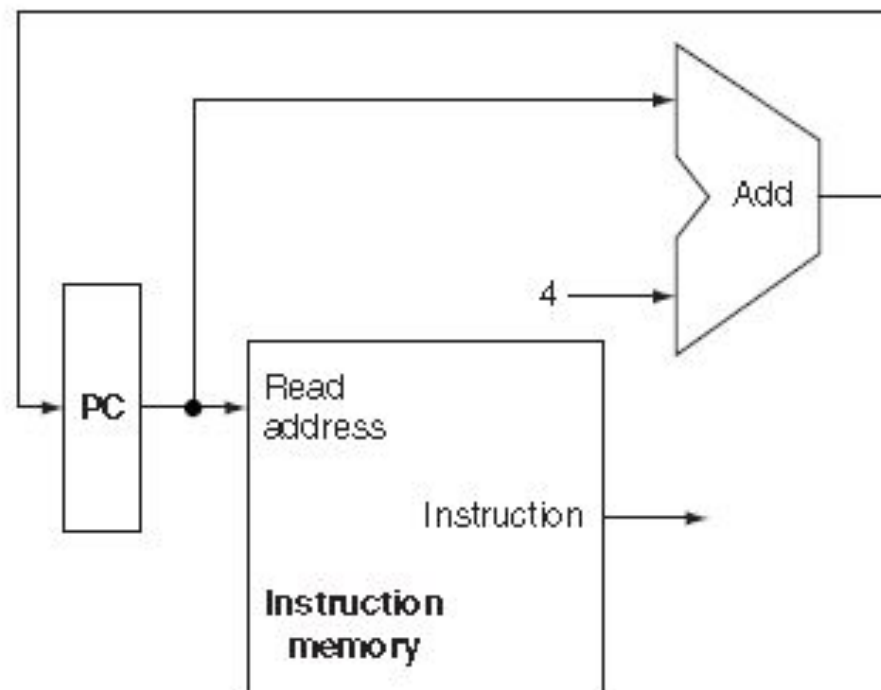
$\$s1$     $\$s2$     $\$t0$   
↑   ↑   ↑  
 $g = h + A[8];$

Correspondente assembly com offset correto:

```
lw $t0, 32($s3) # $t0 recebe A[8]
add $s1, $s2, $t0 # $s1 recebe a soma de h + A[8]
```

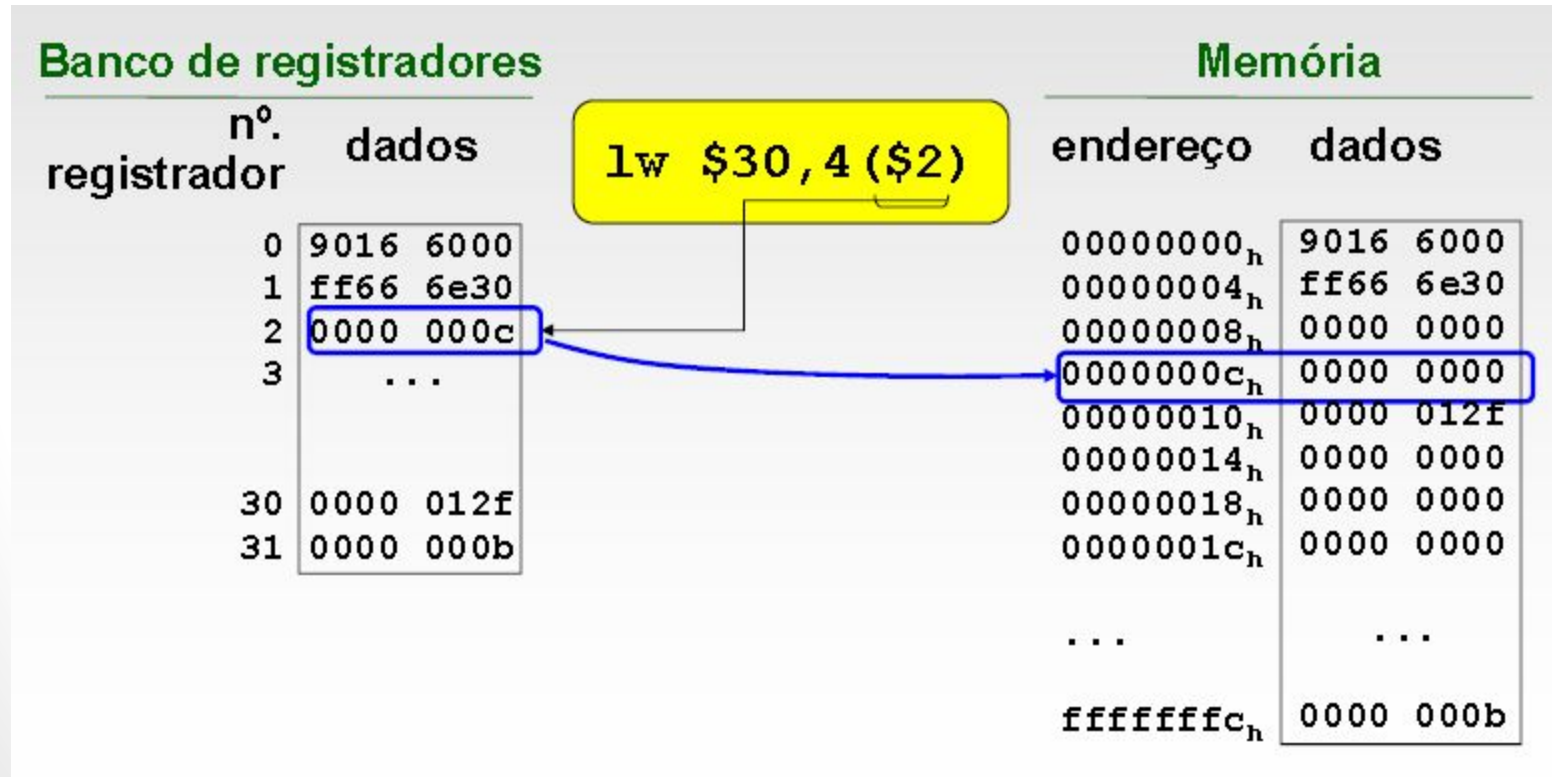
# Memória de Instruções

Para cada nova instrução, o registrador contador de programas (PC – Program Counter) precisa ser somado a 4 (cada instrução ocupa 4 bytes – 4 posições de memória).



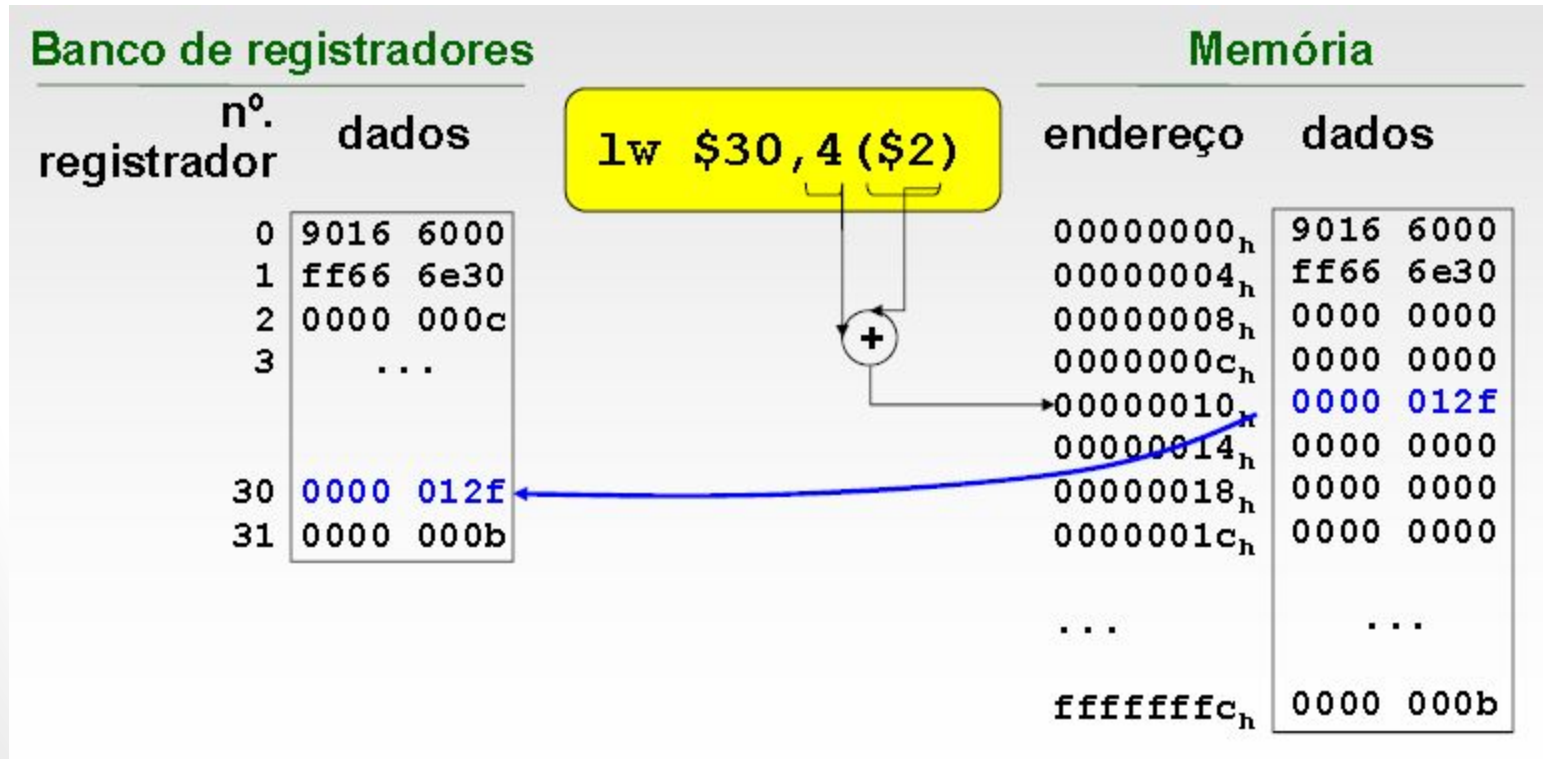
# Instruções de movimentação de dados

## Instrução lw: Load Word



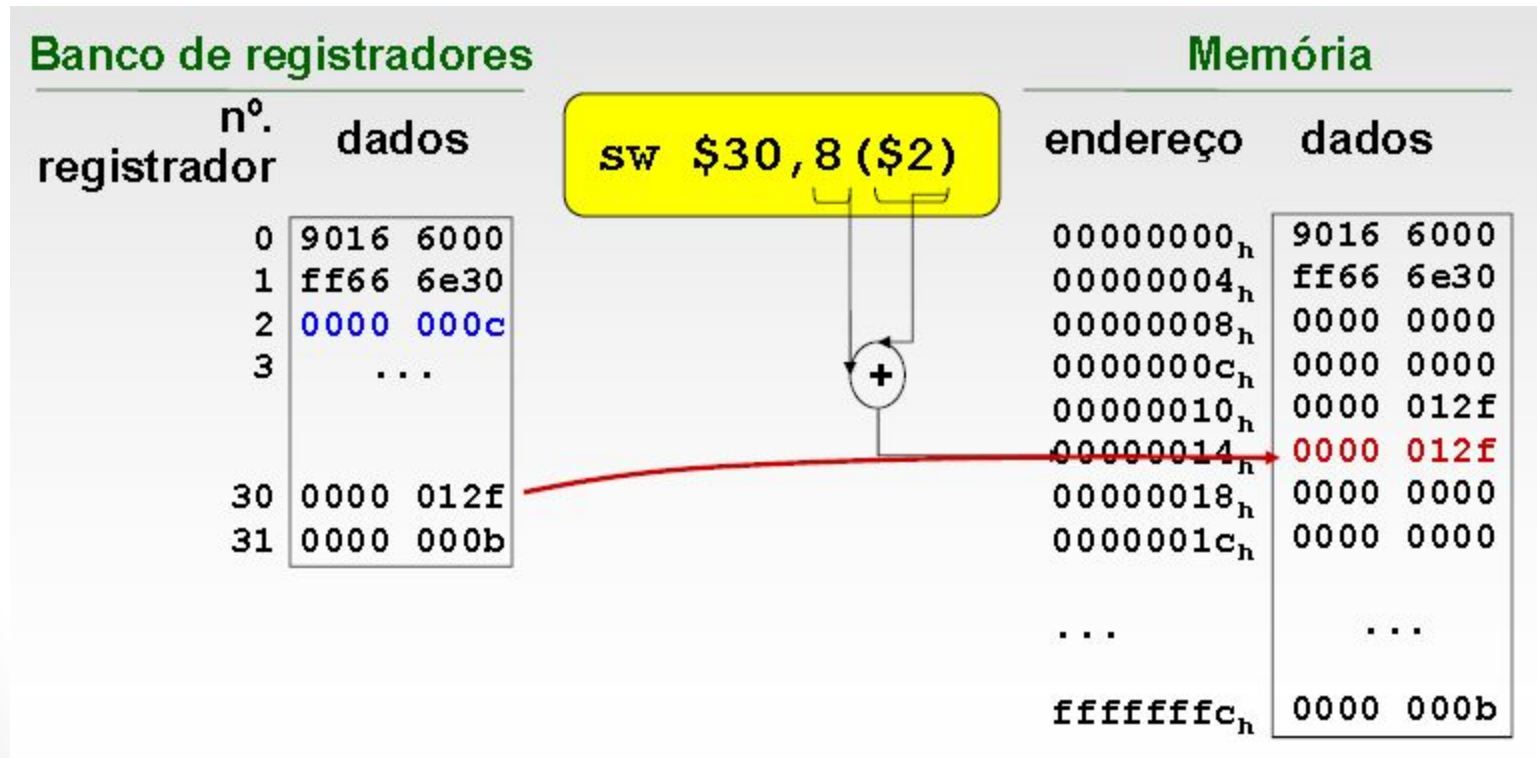
# Instruções de movimentação de dados

## Instrução lw: Load Word



# Instruções de movimentação de dados

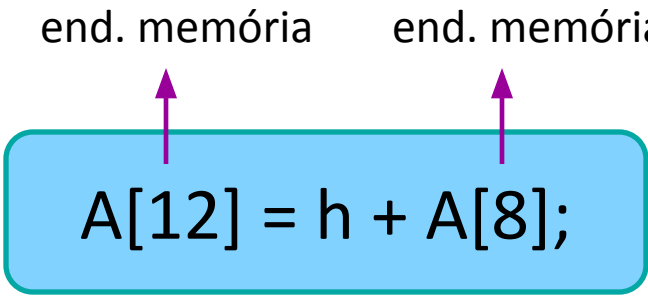
## Instrução sw: Store Word



# Instruções de movimentação de dados

## Código C:

end. memória      end. memória



```
A[12] = h + A[8];
```

- Agora o resultado precisa ser armazenado na memória;
- Nesse caso, usa-se a instrução **sw** = store word;

```
lw $t0, 32($s3) # $t0 recebe A[8]
add $s1, $s2, $t0 # $s1 recebe a soma de h + A[8]
sw $s1, 48($s3)   # armazena h + A[8] no endereço
                  # correspondente a A[12]
```



# Campos das instruções MIPS

## Instruções tipo R – instruções aritméticas

### Formato de instruções tipo R

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

op – operação básica da instrução.

rs – registrador com o 1º operando.

rt – registrador com o 2º operando.

rd – registrador de destino.

shamt – shift amount, não usado em instruções do tipo R.

Funct – junto ao opcode, define a operação.

# Campos das instruções MIPS

## Instruções tipo R – instruções aritméticas

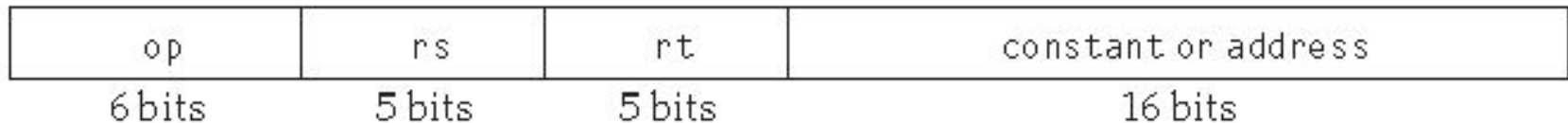
### Formato de instruções tipo R

`add $t0, $s1, $s2`

	op	rs	rt	rd	shamt	funct
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
	<code>add</code>	<code>\$s1</code>	<code>\$s2</code>	<code>\$t0</code>	<code>--</code>	<code>add</code>
Instrução (decimal):	0	17	18	8	0	(20) <sub>h</sub>
	<code>add</code>	<code>\$s1</code>	<code>\$s2</code>	<code>\$t0</code>	<code>--</code>	<code>add</code>
Instrução (binário):	000000	10001	10010	01000	00000	100000

# Campos das instruções MIPS

**Princípio de projeto 3:** Agilize os casos mais comuns. Existe uma outra possibilidade de operações aritméticas, são as instruções imediatas.



op – operação básica da instrução.

rs – registrador com o 1º operando.

**rt – registrador de destino (terceiro campo).**

Constant – constante imediata.

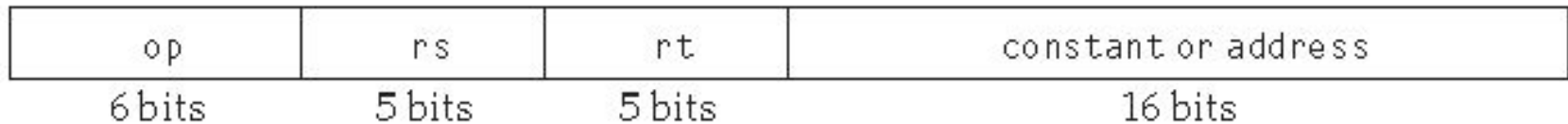
Exemplo:

addi \$s1, \$s2, 100      # \$s1 = \$s2 + 100

# Campos das instruções MIPS

**Princípio de projeto 3:** Agilize os casos mais comuns. Existe uma outra possibilidade de operações aritméticas, são as instruções imediatas.

Instruções do tipo I:



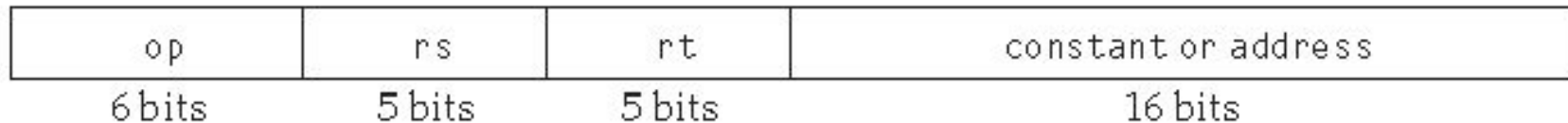
Exemplo 2:

`lw $t0, 32($s3) # registrador $t0 recebe A[8]`

# Campos das instruções MIPS

**Instruções do tipo I** – immediate - instruções imediatas e de transferência de dados.

Não seria possível manter o formato tipo R, que tem apenas 5 bits em cada campo ( $2^5 = 32$ ). Pois o maior valor de endereço admissível seria o 32.



O formato de instruções do tipo I permite carregar uma word dentro de uma região de  $\pm 2^{15} = 32768$  bytes ou  $2^{13} = 8192$  words.

# Campos das instruções MIPS

**Instruções do tipo I** – immediate - instruções imediatas e de transferência de dados.

`addi $s3, $s2, 4      # $s3 ← $s2 + 4`

Instrução (decimal):	op	rs	rt	immediate
	8	18	19	4
	<code>addi</code>	<code>\$s2</code>	<code>\$s3</code>	<code>constante</code>
Instrução (binário):	001000	10010	10011	00000000000000100
	<code>addi</code>	<code>\$s2</code>	<code>\$s3</code>	<code>constante</code>

# Campos das instruções MIPS

**Instruções do tipo I** – immediate - instruções imediatas e de transferência de dados.

`lw $t0, 32 ($s2)`

	op	rs	rt	immediate
	6 bits	5 bits	5 bits	16 bits
	<code>lw</code>	<code>\$s2</code>	<code>\$t0</code>	<code>offset</code>
Instrução (decimal):	(23) <sub>h</sub>	18	8	32
	<code>lw</code>	<code>\$s2</code>	<code>\$t0</code>	<code>offset</code>
Instrução (binário):	<b>100011</b>	<b>10010</b>	<b>01000</b>	<b>0000000000100000</b>

# Instruções MIPS

## Instruções do tipo I

Instrução **lui** – load upper immediate

Carrega constante nos 16 bits mais altos

`lui $t0, 1023`

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits
lui	--	\$t0	immediate

Instrução  
(decimal):

(F) <sub>h</sub>	0	8	1023
lui	--	\$t0	immediate

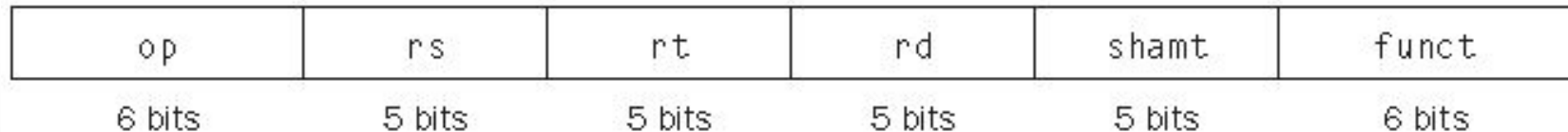
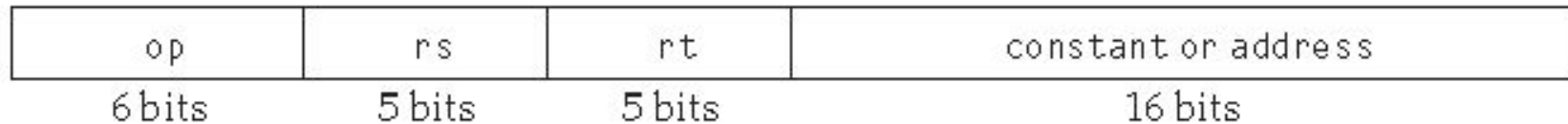
Instrução  
(binário):

001111	00000	01000	0000011111111111
--------	-------	-------	------------------



# Campos das instruções MIPS

Ainda assim, é possível manter os 3 primeiros campos idênticos entre estes 2 formatos:



# Instruções MIPS

Instruções vistas até aqui:

Instrução	Formato	rs	rt	rd	shamt	funct	endereço
add	R	reg	reg	reg	0	32 <sub>dec</sub>	n.a
sub	R	reg	reg	reg	0	34 <sub>dec</sub>	n.a
add immediate	I	reg	reg	n.a.	n.a.	n.a	constante
lw (load word)	I	reg	reg	n.a.	n.a.	n.a	endereço
sw (store word)	I	reg	reg	n.a.	n.a	n.a	endereço

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	Three operands; data in registers
	subtract	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	Three operands; data in registers
	add immediate	addi \$s1,\$s2,100	$\$s1 = \$s2 + 100$	Used to add constants
Data transfer	load word	lw \$s1,100(\$s2)	$\$s1 = \text{Memory}[\$s2 + 100]$	Data from memory to register
	store word	sw \$s1,100(\$s2)	$\text{Memory}[\$s2 + 100] = \$s1$	Data from register to memory

# Instruções MIPS

## Linguagem de máquina do MIPS:

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add \$s1,\$s2,\$s3
sub	R	0	18	19	17	0	34	sub \$s1,\$s2,\$s3
addi	I	8	18	17	100			addi \$s1,\$s2,100
lw	I	35	18	17	100			lw \$s1,100(\$s2)
sw	I	43	18	17	100			sw \$s1,100(\$s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

# Instruções MIPS

Exercício 1: Obter os assemblys para os códigos C:

```
c = 20 + A[10];  
d = b + c + d;  
e = 100 + d;
```



# ORCJogo

Obter os assemblys para os códigos  
C:

```
n = 20 + x[5];  
z = y + m + n;  
p = z + 85;
```

# Instruções MIPS

- Exercício 2: Suponha que o endereço base da matriz B esteja armazenado em \$s4.

Qual o código assembly para trocar os valores do B[10] e do B[11]?

# Instruções MIPS

- Instruções de desvio condicional

Instruções de desvio condicional:

Exemplo:

**bne \$s0, \$s1, fim**      # vai para fim se \$s0 ≠ \$s1

- **beq registrador1, registrador2, L1**
  - se o valor do registrador1 for igual ao do registrador2 o programa será desviado para o label L1 ( beq = branch if equal).
- **bne registrador1, registrador2, L1**
  - se o valor do registrador1 não for igual ao do registrador2 o programa será desviado para o label L1 ( beq = branch if not equal).

# Instruções MIPS

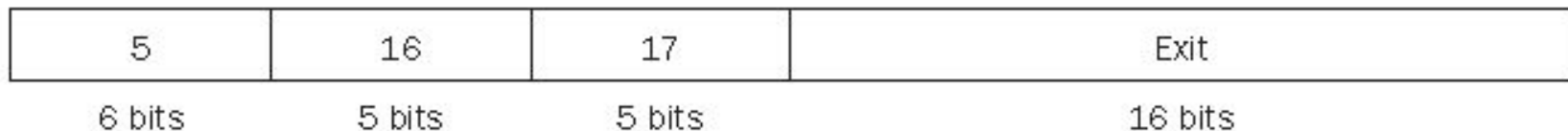
- Instruções de desvio condicional

Instruções de desvio condicional:

Exemplo:

**bne \$s0, \$s1, fim**      # vai para fim se \$s0  $\neq$  \$s1

Formato da instrução:



O endereço ficaria limitado em apenas 16 bits (endereços de  $2^{16}$ ). Dessa forma, o endereço é calculado da seguinte maneira:

**PC = Reg + Endereço de desvio (16 bits da instrução)**



# Instruções MIPS

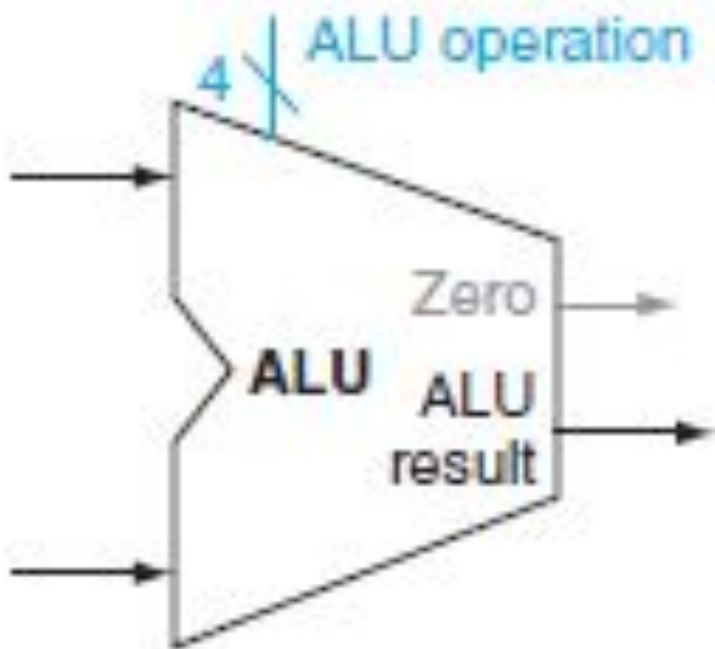
## Instruções de desvio condicional

**PC = PC + Endereço de desvio (16 bits da instrução)**

- Os desvios a partir do PC permitem desviar dentro de  $\pm 2^{15}$ .
- A decisão de usar o PC para a soma do endereço de desvio se deve pelo fato de **a maioria dos desvios irem para locais a poucas instruções de distância.**
- Essa forma de desvio é chamada de **endereçamento relativo ao PC.**

# ALU / ULA

**beq \$s0, \$s1, fim** # vai para fim se \$s0 = \$s1



- A ALU possui um bit de saída que indica se o resultado é zero.
- É utilizado em instruções de **branch**.
- A comparação é feita subtraindo os operandos.

# Instruções MIPS

- **Instruções formato tipo J – jump (desvios)**

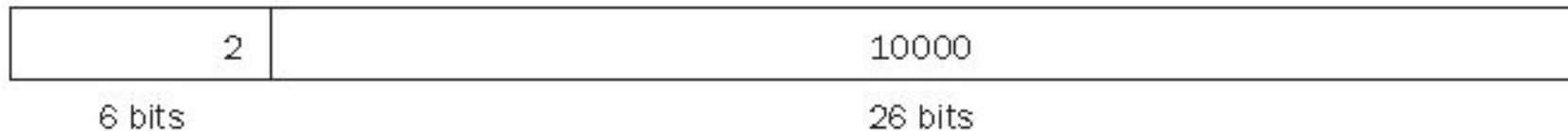
Consistem de 6 bits para o campo da operação e 26 bits para o endereço.

Exemplo:

**J 10000** # vai para a posição 40000

O código da operação jump é 2.

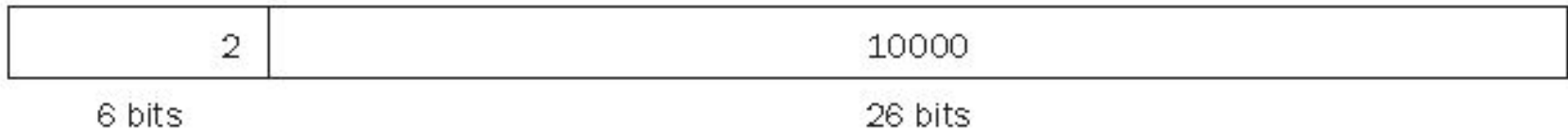
Formato da instrução:



# Instruções MIPS

- Instruções formato tipo J – jump (desvios)

Formato da instrução:



Os 26 bits são deslocados 2x, formando 28 bits.

**Porque esse deslocamento é feito?**

- Com os 28 bits o desvio é por instrução e não por byte.

**De onde vem os 4 bits para formar os 32 bits do endereço?**

- 4 bits precisam ainda complementar os 28 bits da instrução a fim de completar os 32 bits. Estes bits são formados com os 4 bits mais significativos do PC.

# Instruções do MIPS

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	Three register operands
	subtract	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	Three register operands
	add immediate	addi \$s1,\$s2,100	$\$s1 = \$s2 + 100$	Used to add constants
Data transfer	load word	lw \$s1,100(\$s2)	$\$s1 = \text{Memory}[\$s2 + 100]$	Word from memory to register
	store word	sw \$s1,100(\$s2)	$\text{Memory}[\$s2 + 100] = \$s1$	Word from register to memory
	load half	lh \$s1,100(\$s2)	$\$s1 = \text{Memory}[\$s2 + 100]$	Halfword memory to register
	store half	sh \$s1,100(\$s2)	$\text{Memory}[\$s2 + 100] = \$s1$	Halfword register to memory
	load byte	lb \$s1,100(\$s2)	$\$s1 = \text{Memory}[\$s2 + 100]$	Byte from memory to register
	store byte	sb \$s1,100(\$s2)	$\text{Memory}[\$s2 + 100] = \$s1$	Byte from register to memory
	load upper immed.	lui \$s1,100	$\$s1 = 100 * 2^{16}$	Loads constant in upper 16 bits
Logical	and	and \$s1,\$s2,\$s3	$\$s1 = \$s2 \& \$s3$	Three reg. operands; bit-by-bit AND
	or	or \$s1,\$s2,\$s3	$\$s1 = \$s2   \$s3$	Three reg. operands; bit-by-bit OR
	nor	nor \$s1,\$s2,\$s3	$\$s1 = \sim (\$s2   \$s3)$	Three reg. operands; bit-by-bit NOR
	and immediate	andi \$s1,\$s2,100	$\$s1 = \$s2 \& 100$	Bit-by-bit AND reg with constant
	or immediate	ori \$s1,\$s2,100	$\$s1 = \$s2   100$	Bit-by-bit OR reg with constant
	shift left logical	sll \$s1,\$s2,10	$\$s1 = \$s2 \ll 10$	Shift left by constant
	shift right logical	srl \$s1,\$s2,10	$\$s1 = \$s2 \gg 10$	Shift right by constant
Conditional branch	branch on equal	beq \$s1,\$s2,25	if ( $\$s1 == \$s2$ ) go to $PC + 4 + 100$	Equal test; PC-relative branch
	branch on not equal	bne \$s1,\$s2,25	if ( $\$s1 \neq \$s2$ ) go to $PC + 4 + 100$	Not equal test; PC-relative
	set on less than	slt \$s1,\$s2,\$s3	if ( $\$s2 < \$s3$ ) $\$s1 = 1$ ; else $\$s1 = 0$	Compare less than; for beq, bne
	set less than immediate	slti \$s1,\$s2,100	if ( $\$s2 < 100$ ) $\$s1 = 1$ ; else $\$s1 = 0$	Compare less than constant
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr \$ra	go to \$ra	For switch, procedure return
	jump and link	jal 2500	$\$ra = PC + 4$ ; go to 10000	For procedure call

# Instruções MIPS

- Exercício 3: Para o código C abaixo, qual o assembly MIPS:

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

# Instruções de desvio incondicional

Exercício 4:

```
while (j != k) {  
    j++;  
    a = a - b;  
}
```

Considere: \$s2 para  $j$ , \$s3 para  $k$ , \$s4 para  $a$  e \$s5 para  $b$

# Instruções MIPS

## Instruções de Operações Lógicas

Algumas destas instruções operam bit a bit sobre os dados. Elas são úteis no empacotamento e desempacotamento de palavras.

Operações lógicas	Operadores C	Operadores Java	Instruções 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 MIPS

## Instruções de Operações Lógicas

As instruções de *shifts* (deslocamentos) movem todos os bits de uma palavra para a esquerda ou para a direita, preenchendo os bits vazios com zeros (0).

sll – shift left logical

srl – shift right logical

Exemplos:

sll \$t2, \$s0, 4

# registrador \$t2 recebe o valor do registrador \$s0 deslocado em 4 posições.

# Instruções MIPS

## Instruções de Operações Lógicas

Exemplos:

**sll \$t2, \$s0, 4**

Se o valor do registrador \$s0 for 9 em decimal:

0000 0000 0000 0000 0000 0000 0000 1001<sub>bin</sub>

Com o deslocamento de 4 bits para a esquerda, o valor passa a ser 144, veja:

0000 0000 0000 0000 0000 0000 0000 1001 0000<sub>bin</sub>

# Instruções MIPS

## Instruções de Operações Lógicas

As instruções de deslocamentos ainda são úteis para realizar a multiplicação ou divisão de um número de forma muito rápida.

O deslocamento a esquerda de  $i$  bits equivale a multiplicar por  $2^i$ .

No exemplo anterior (`sll $t2, $s0, 4`) o valor de `$s0` é multiplicado por 16 ( $9 \times 16 = 144$ ).

$$16 = 2^4$$

# Instruções MIPS

## Instruções de Operações Lógicas

O campo **shamt**, até então não utilizados para as demais instruções, é utilizado para as instruções de deslocamento para definir o número de bits a serem deslocados.

Exemplo:

oprs	rt	rd	shamt	funct	
0	0	16	10	4	0

# Instruções MIPS

## Instruções de Operações Lógicas - AND

As instruções AND, OR e NOT são instruções que operam bit a bit.

Exemplo: **and \$t0, \$t1, \$t2**

0000 0000 0001 0010 0111 0101 1111 1110 - \$t1

0000 0001 0111 0111 1111 0000 1010 1011 - \$t2

---

**0000 0000 0001 0010 0111 0000 1010 1010 - \$t0**

# Instruções MIPS

## Instruções de Operações Lógicas - OR

As instruções AND, OR e NOT são instruções que operam bit a bit.

Exemplo: **or \$t0, \$t1, \$t2**

0000 0000 0001 0010 0111 0101 1111 1110 - \$t1

0000 0001 0111 0111 1111 0000 1010 1011 - \$t2

---

**0000 0001 0111 0111 1111 0101 1111 1111 - \$t0**

# Instruções MIPS

## Instruções de Operações Lógicas - NOR

Nesse caso, o formato das instruções foi mantido e, dessa forma a instrução utiliza dois registradores. A instrução foi chamada de *nor* (*not or*). Para fazer *not* basta usar um registrador com valor zero.

Exemplo: **nor \$t0, \$t1, \$t3**

0000 0000 0000 0000 0000 0000 0000 0000 - \$t1

0000 0001 0111 0111 1111 0000 1010 1011 - \$t3

---

**1111 1110 1000 1000 0000 1111 0101 0100 - \$t0**

# Instruções MIPS

## Instruções de Operações Lógicas

Além das instruções lógicas que operam sobre os registradores, as instruções de AND e OR podem ser realizadas com um operando imediato:

Exemplos:

`andi $s1, $s2, 100`    #  $\$s1 = \$s2 \& 100$

`ori $s1, $s2, 100`    #  $\$s1 = \$s2 | 100$



# Instruções MIPS

- Além dos testes de igualdade e desigualdade, é também útil o teste para verificar qual a maior/menor variável.

No MIPS essa verificação é feita com as instruções:

**slt** - set on less than

**slti** - set on less than immediate

**slt** - compara 2 registradores e atribui 1 a um terceiro registrador.

**slti** – compara o valor de um registrador com um operando imediato e atribui 1 no segundo registrador.

# Instruções MIPS

Exemplos:

`slt $t0, $s3, $s4 # $t0 = 1 se  $\$s3 < \$s4$`

Se o valor do registrador \$s3 for menor do que \$s4, \$t0 recebe 1, senão \$t0 recebe 0.

`slt $t0, $s3, $20 # $t0 = 1 se  $\$s3 < 20$`

Se o valor do registrador \$s3 for menor do que 20 , \$t0 recebe 1, senão \$t0 recebe 0.

# Instruções MIPS

- Exercício 5: Para o código C abaixo, qual o assembly MIPS:

```
if (i >= j) {  
    f = f * h;  
else  
    h --;  
}
```

# Instruções MIPS

- Exercício 6: Para o código C abaixo, qual o assembly MIPS:

```
for (i = 0; i < x; i = i + 1) {  
    y = y + 1;  
}
```

# Instruções MIPS

- Exercício 7: Para o código C abaixo, qual o assembly MIPS:

```
for (i = 0; i <= 100; i = i + 1) {  
    a = i + c;  
}
```

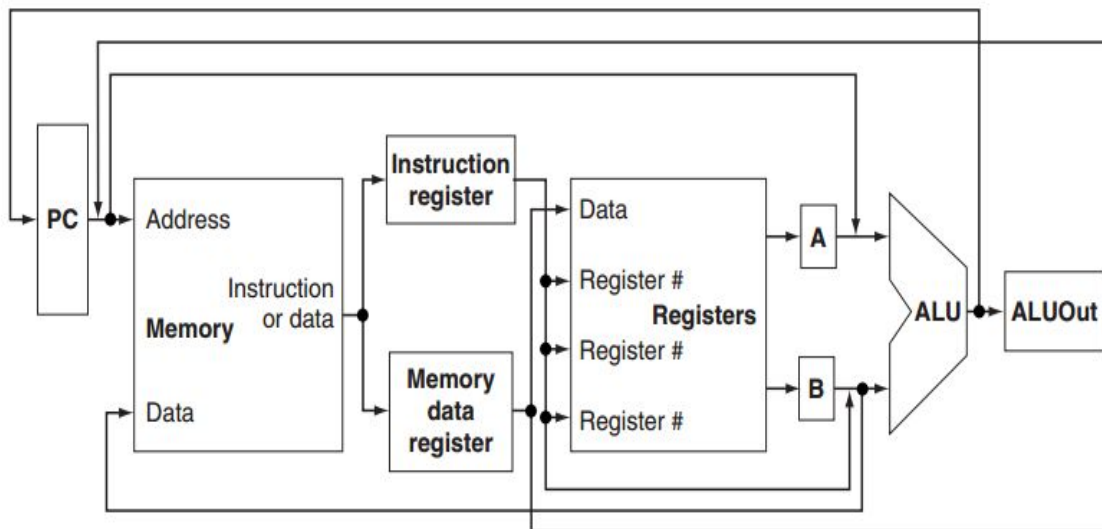
# **Caminho de dados da organização do MIPS**

# Caminho de dados

Registradores acrescentados na implementação multiciclo:

- Registrador de Instrução – Instruction Register (RI)
- Registrador de dados da memória – Memory Data Register (MDR).

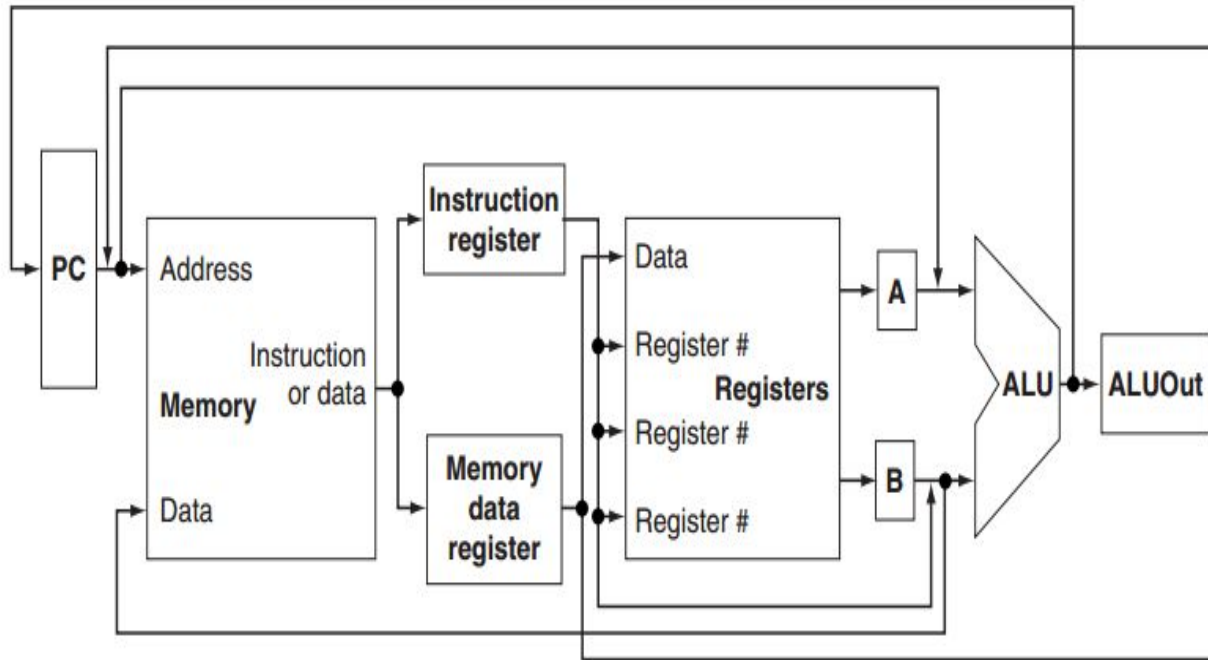
RI e MDR são usados para salvar a saída da memória referentes a uma leitura de instrução e uma leitura de dados, respectivamente.



# Caminho de dados

Registradores acrescentados na implementação multiciclo:

- **Registradores A e B** usados para conter os valores dos registradores operandos lidos do banco de registradores.
- **Registrador ALUOut** – registrador de saída da ALU





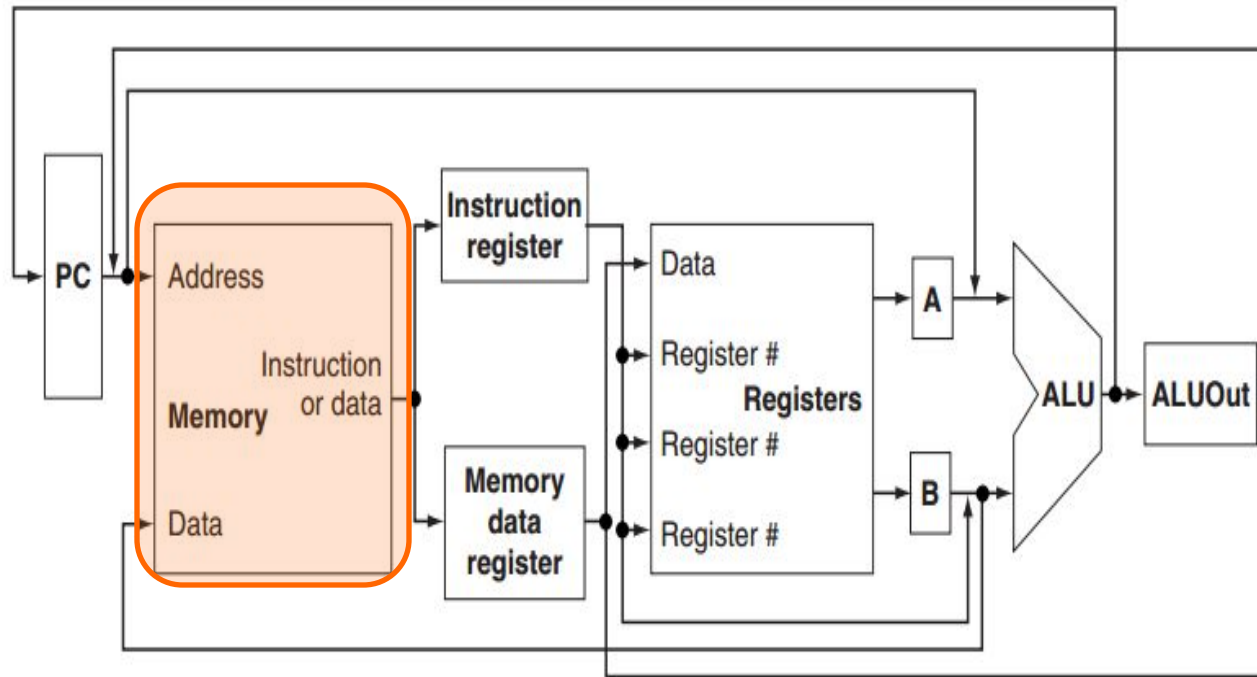
# Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

`add $s3, $s2, $s1`

1º) Busca da instrução na memória de instruções:

PC contém o endereço da próxima instrução. A partir desse endereço a instrução é lida e salva no RI.

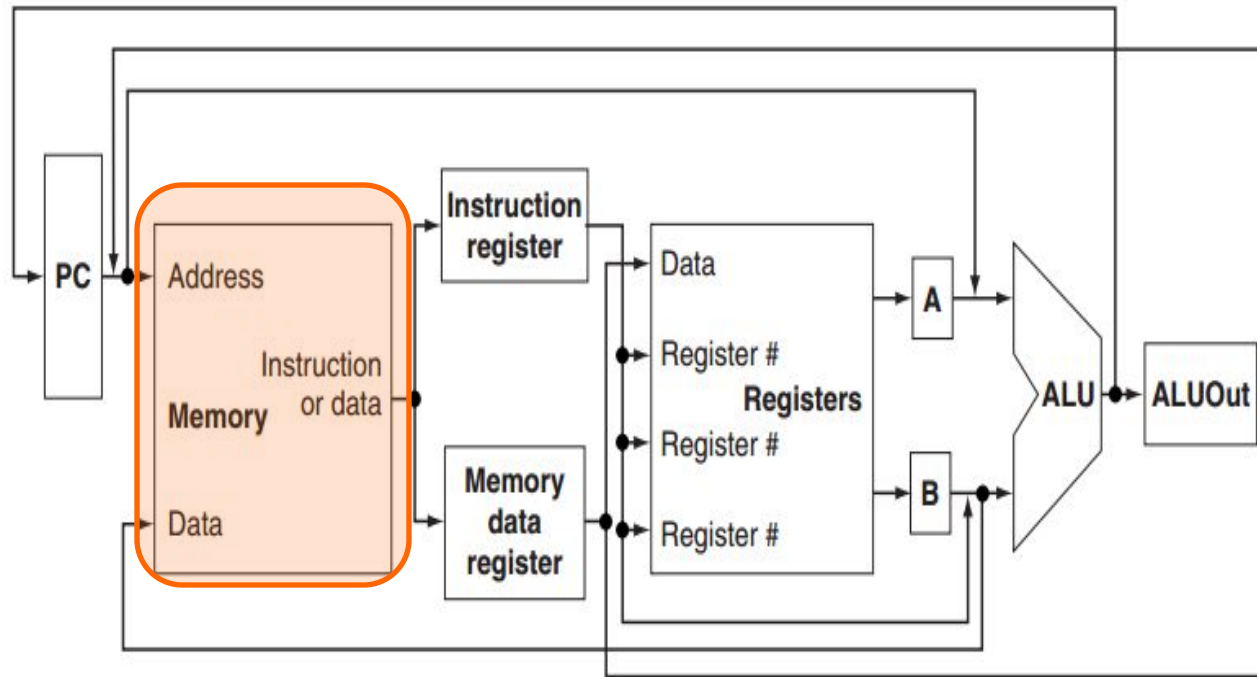


# Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

`add $s3, $s2, $s1`

2º) Próximo passo, a instrução é decodificada. Nesse caso é verificada a operação a ser realizada de forma a ativar a operação e os módulos requeridos em cada ciclo pela unidade de controle.

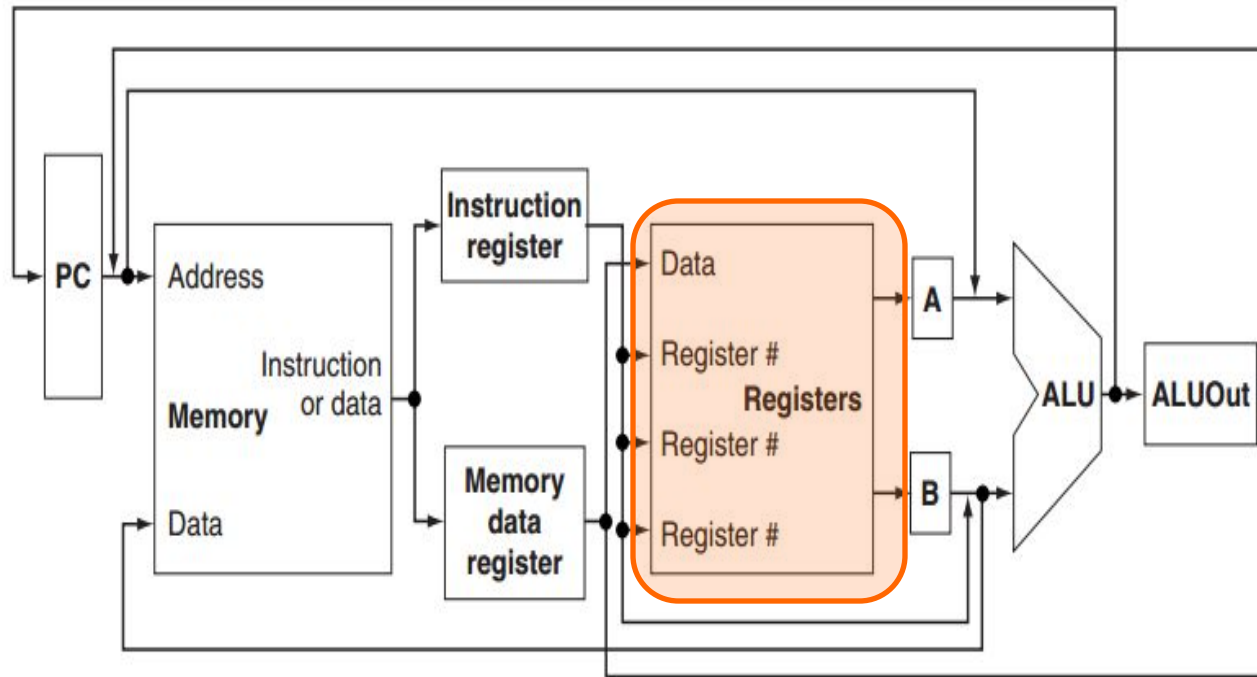


# Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

`add $s3, $s2, $s1`

3º) Como é uma instrução do tipo R, os operandos estão no banco de registradores. Os campos referentes aos registradores usados são informados e os dados a serem somados são lidos e escritos nos registradores A e B.

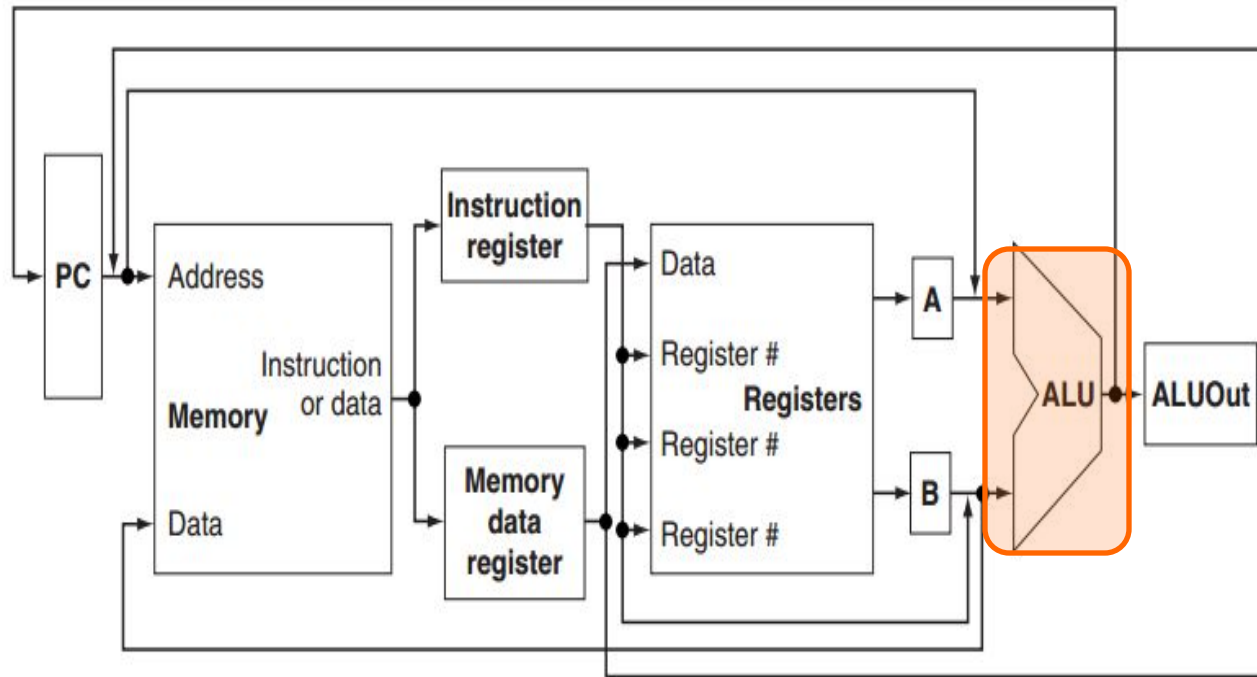


# Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

`add $s3, $s2, $s1`

4º) A soma entre os registradores A e B (referentes aos registradores \$s1 e \$s2) é realizada na ALU e o resultado é salvo no registrador de saída da ALU (ALUOut).



# Caminho de dados

Exemplo do caminho de dados da seguinte instrução:

`add $s3, $s2, $s1`

5º) O resultado da operação precisa ser escrito no registrador de escrita do banco de registradores, nesse caso, o \$s3. Para isso, o registrador de escrita é identificado na instrução (no IR) e o resultado da soma escrito.

