Arquitetura de Computadores

Unichristus
Centro Universitário Christus

Unidade 03: Instruções: a Linguagem de Máquina

Daniel Teixeira

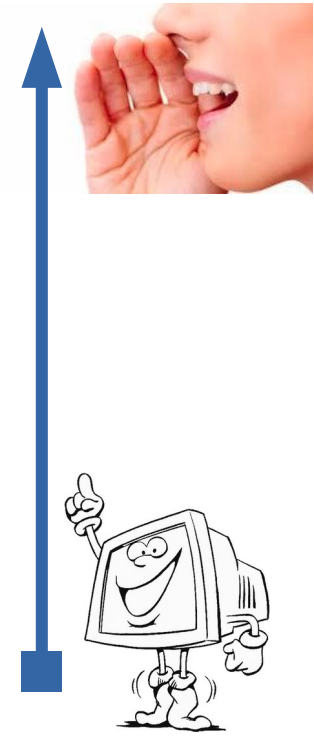
prof.danielnt@gmail.com
dant25.github.io/professor

Motivação

- Linguagem
  - Maneira de exprimir-se própria de um povo, fala, dialeto, língua.
  - Conjunto de símbolos, palavras e regras na construção de sentenças que expressam e processam instruções para computadores.

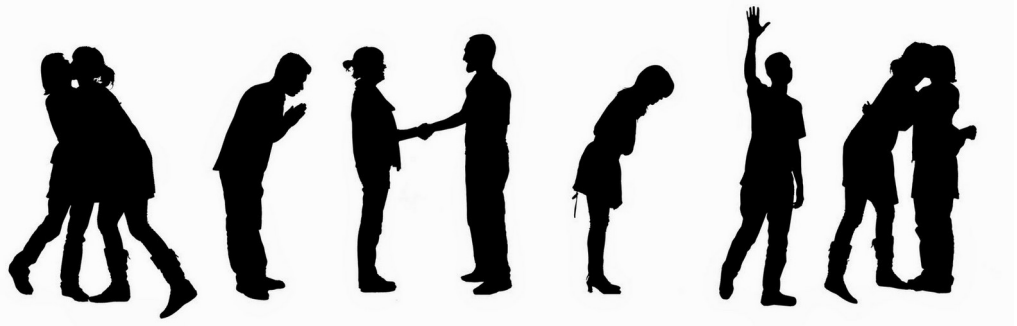
Motivação

- Linguagem de **alto nível**
  - Essas são aquelas cuja sintaxe se aproxima mais da nossa linguagem humana
- Linguagem de **baixo nível**
  - É aquela que se aproxima mais da linguagem de máquina.

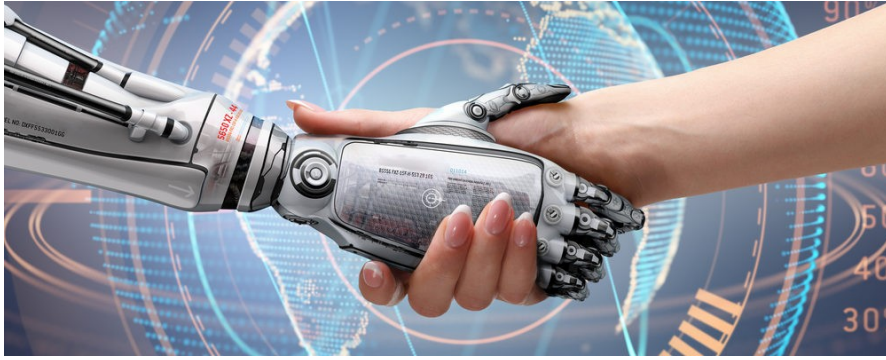


Motivação

- Para controlar o hardware de um computador é preciso falar a sua **linguagem**
- As palavras da linguagem de um computador são chamadas **instruções**
- O seu vocabulário é chamado de **conjunto de instruções**

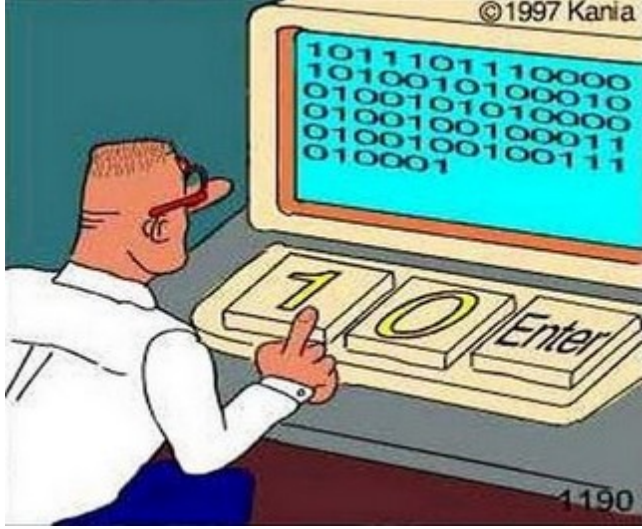


Motivação



- Linguagens de Computador x Humanos
  - Ambas possuem grande diversidade
    - Maquinas: seriam como se fosse dialetos de uma linguagem
      - Se aprender uma, será mais fácil aprender as outras
    - Humanos: Além de existir os dialetos existem mais linguagens
      - Muito mais difícil aprender as outras

Motivação



- Linguagens de Computador
  - Por que essas linguagens são parecidas?
    - Operações básicas: todas precisam oferecer
    - Tecnologias de hardware são baseadas em princípios semelhantes
    - Todas procuram:
      - Facilitar o projeto do hardware
      - Maximizar desempenho
      - Minimizar custo

Motivação

- Conjunto de instrução usadas será do **MIPS**
  - **M**icroprocessador sem e**S**tágios **I**ntertravados de **P**ipeline (**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages)
  - Tecnologia **RISC**
  - Não confundir com **M**ilhões de **I**nstruções **P**or **S**egundo

Operações do Hardware do Computador

- Instruções que todo computador precisa ter são as **Operações Aritméticas**
- **add a, b, c**

  a = b + c

  - Soma **b** e **c** e coloca o resultado em **a**
- **sub a, b, c**

  a = b - c

  - Subtrai **b** e **c** e coloca o resultado em **a**

Operações do Hardware do Computador

- Exemplo:
  - Somar as variáveis **b**, **c**, **d** e **e**.

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

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

Operações do Hardware do Computador

- Cada linha contém apenas uma instrução
- # é o símbolo para fazer comentários

```
add a, b, c     # A soma b + c é colocada em a.
add a, a, d     # A soma b + c + d agora está em a.
add a, a, e     # A soma b + c + d + e agora está em a.
```

Operações do Hardware do Computador

- Toda instrução deve conter **exatamente 3** operações
- **add a, b, c**
  - **2** números somados e **1** local para armazenar
- Hardware com um conjunto de instrução variável é mais complexo que um de quantidade fixa

- **Princípio de projeto 01**:
  - Simplicidade favorece a regularidade

Operações do Hardware do Computador

- Como seria a tradução das linhas abaixo para assembly do MIPS?

  a = b + c;

  d = a – e;

Operações do Hardware do Computador

- Como seria a tradução das linhas abaixo para assembly do MIPS?

  a = b + c;

  d = a – e;

add a, b, c
sub d, a, e

Operações do Hardware do Computador

- Como seria a tradução da linha abaixo para assembly do MIPS?

  f = (g+h) – (i+j);

Operações do Hardware do Computador

- Como seria a tradução da linha abaixo para assembly do MIPS?

  f = (g+h) – (i+j);

  add a, g, h
  add b, i, j
  sub f, a, b

- Temos 5 variáveis na expressão (**f**,**g**,**h**,**i** e **j**)
- Não existem as variáveis **a** e **b**
  - Se existisse, poderiam estar sendo usadas e iriamos acabar perdendo os valores anteriores delas

Operações do Hardware do Computador

- Apenas 1 operação é feita por instrução MIPS
- Para somar (g+h) temos que colocar o resultado em algum lugar
  - Variáveis temporárias: t0, t1, t2, ...

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)
```

Operações do Hardware do Computador

Para determinada função, que linguagem de programação provavelmente utiliza mais linhas de código? Coloque as três representações a seguir em ordem.

1. Java
2. C
3. Assembly do MIPS

Operandos do Hardware do Computador

- Os operandos das instruções são restritos
  - Precisam ser de um grupo limitado de locais especiais no hardware
  - Utiliza-se os **Registradores**
- Registradores MIPS
  - Tamanho: 32 bits (32 bits = 1 palavra)
  - **Possui ao todo 32 registradores**

Operandos do Hardware do Computador

- **Princípio de Projeto 02:**
  - Menor significa mais rápido
- Acessar uma posição de memória maior significa “andar” mais para encontrá-la
  - Mais clocks serão necessários
- 32 registradores precisam de 5 bits de endereçamento → 2^5
  - Se fossem 33 ou 40 registradores, qual seria o problema?

Operandos do Hardware do Computador

- Notação dos registradores das variáveis do programa:
  - $s0, $s1, $s2, ...
- Notação dos registradores temporários
  - $t0, $t1, $t2, ...

Operandos do Hardware do Computador

- Sabemos que **f**, **g**, **h**, **i** e **j** estão associadas aos registradores $s0, $s1, $s2, $s3 e $s4. Como seria a tradução da linha abaixo para assembly do MIPS?

  f = (g+h) – (i+j);

Operandos do Hardware do Computador

- Sabemos que **f**, **g**, **h**, **i** e **j** estão associadas aos registradores $s0, $s1, $s2, $s3 e $s4. Como seria a tradução da linha abaixo para assembly do MIPS?

  f = (g+h) – (i+j);

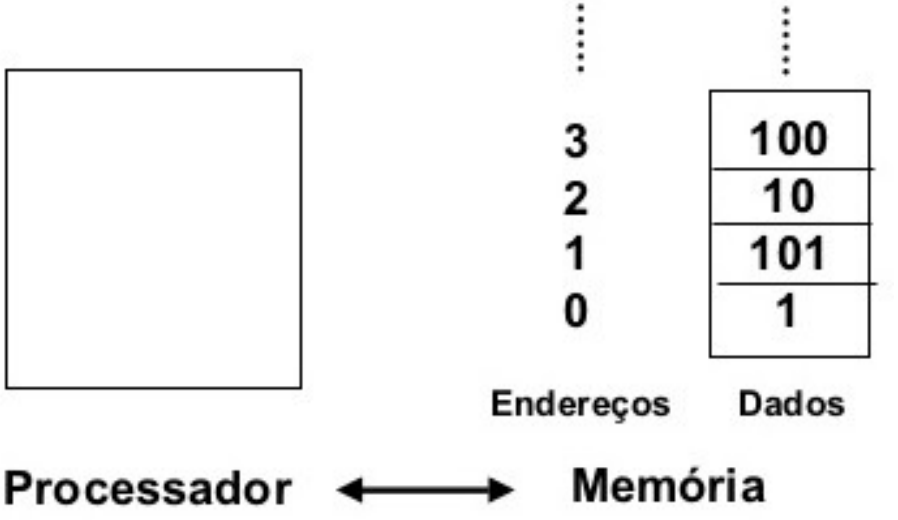
```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

Operandos do Hardware do Computador

- Um programa pode ter mais de 32 variáveis
  - Como um computador consegue representar e acessar essas estruturas?
  - Resposta = Memória RAM
- Todas operações aritméticas devem ser feitas com uso dos registradores
- É necessário transferir dados entre a RAM e os registradores

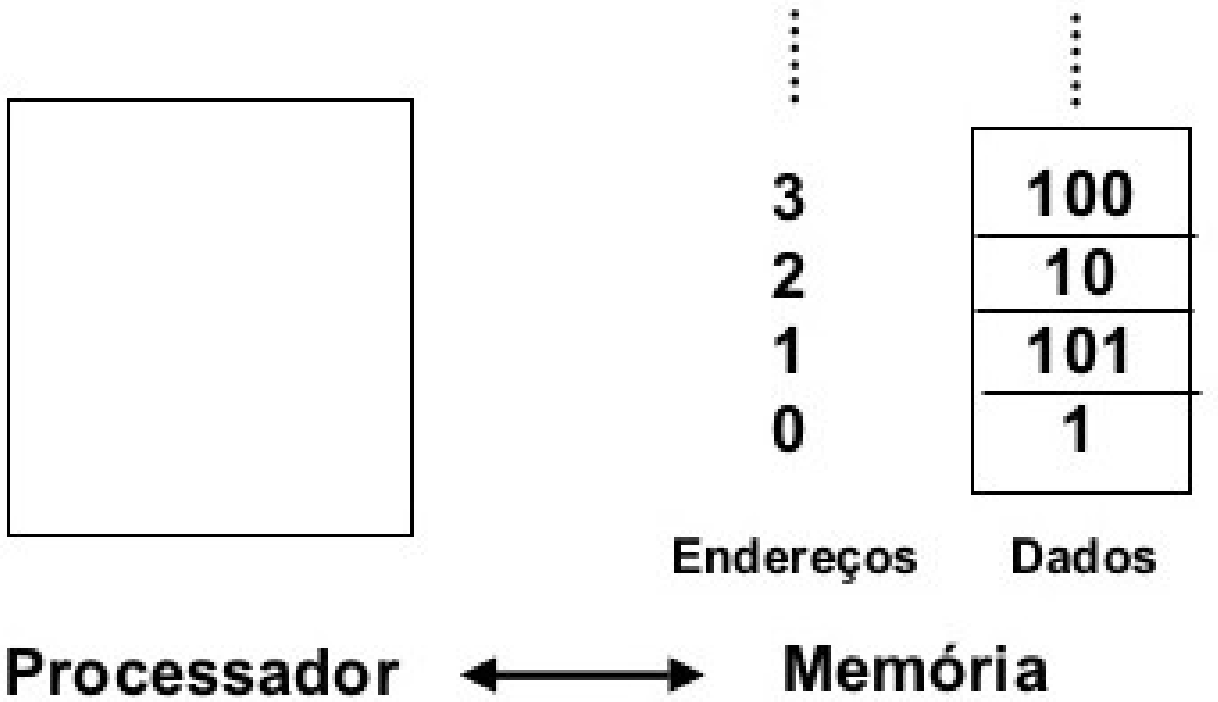
Operandos do Hardware do Computador

- Instrução de transferência de dados
  - Acessa uma palavra (word) na RAM
  - É preciso informar o endereço onde a palavra está
- Memória RAM
  - Sequencia unidimensional



Operandos do Hardware do Computador

- Qual o valor que se encontra no endereço 2?
  - Resposta = 10
- O acesso dos dados se dá através dos endereços



Operandos do Hardware do Computador

- Instrução que copia uma palavra da memória para um registrador
  - Load word → lw
- O formato da instrução **lw**
  - 1º argumento: registrador a ser carregado
  - 2º argumento: uma constante (**offset**)
  - 3º argumento: registrador usado para acessar a memória (**registrador base**)
- Obs: Somando o 2º e 3º argumento dá o endereço
de memória desejado

Operandos do Hardware do Computador

- Vamos supor que A seja uma sequencia de 100 words e que o compilador tenha associado as variáveis g e h aos registradores **$s1** e **$s2**, como antes. Vamos supor também que o endereço inicial da sequencia, ou endereço base, esteja em **$s3**. Compile esta instrução de atribuição:
- g = h + A[8];

Operandos do Hardware do Computador

$s1 $s2 $s3

g = h + A[8];

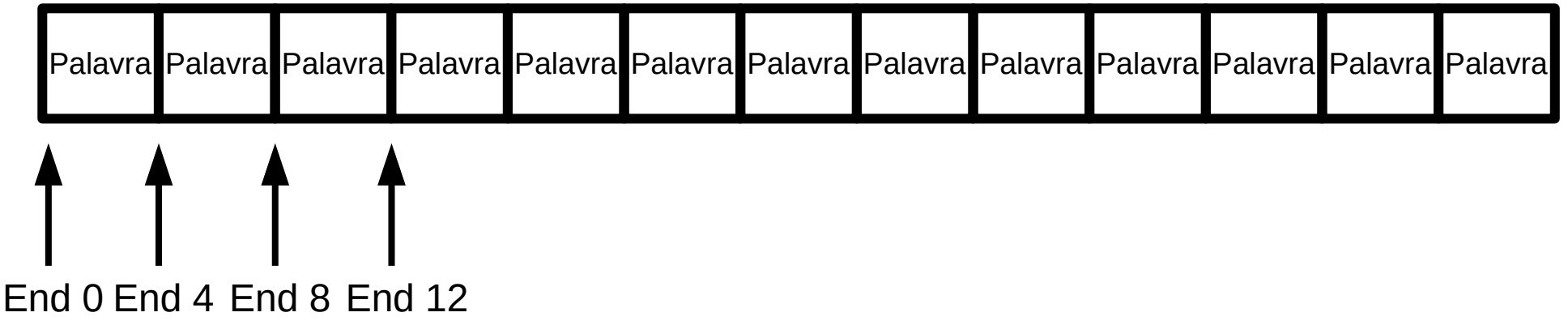
- Copiar A[8] para um registrador temporário
  - **lw $t0, 8($s3)**
- Realizar uma operação aritmética normal
  - **add $s1, $s2, $t0**

Operandos do Hardware do Computador

- O compilador é responsável por alocar todos os dados na memória
  - Ele guarda a referencia para o **endereço onde começa** o dado
- Uma palavra tem 32 bits → 2^5 bits → 4 bytes
  - A cada **32 bits** começa uma nova palavra
  - A cada **4 bytes** começa uma nova palavra

Operandos do Hardware do Computador

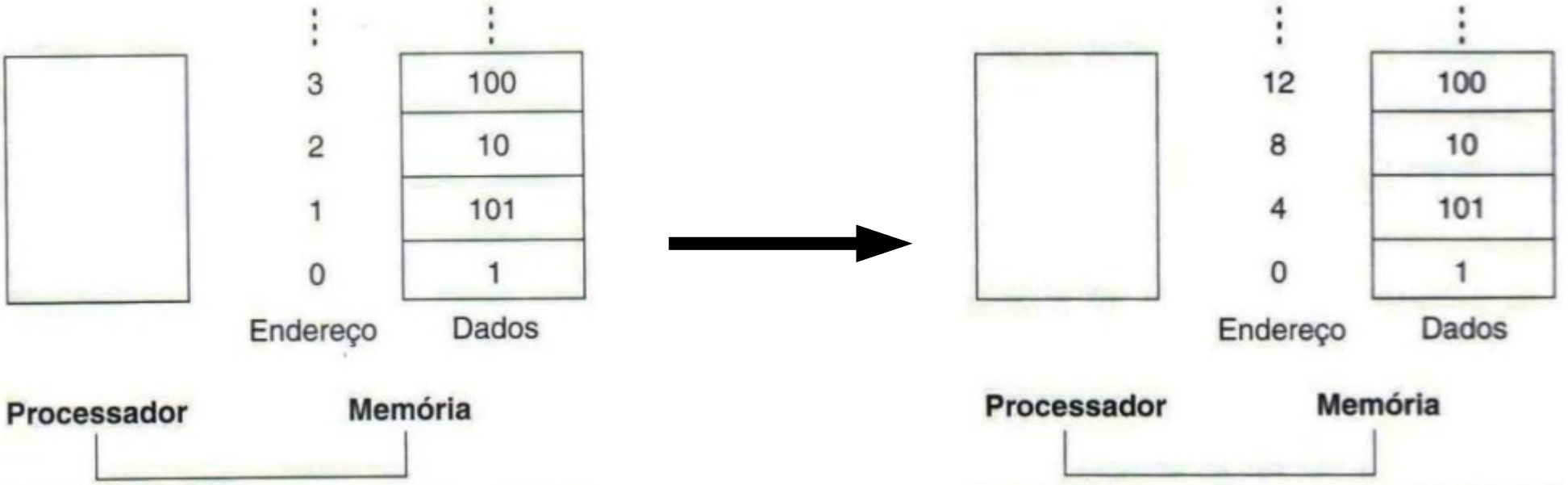
- Referencia para o **endereço onde começa** o dado
- A cada **32 bits** começa uma nova palavra
  - A cada **4 bytes** começa uma nova palavra



A “distancia” entre 2 palavras consecutivas é 4 bytes

Operandos do Hardware do Computador

- Corrigindo



- Qual o valor que se encontra no endereço 2?
  - Resposta = error! Endereço inválido

Operandos do Hardware do Computador

- Vamos supor que A seja uma sequencia de 100 words e que o compilador tenha associado as variáveis g e h aos registradores **$s1** e **$s2**, como antes. Vamos supor também que o endereço inicial da sequencia, ou endereço base, esteja em **$s3**. Compile esta instrução de atribuição:
- g = h + A[8];
  - **lw $t0, 8($s3)**
  - **add $s1, $s2, $t0**

Operandos do Hardware do Computador

- Vamos supor que A seja uma sequencia de 100 words e que o compilador tenha associado as variáveis g e h aos registradores **$s1** e **$s2**, como antes. Vamos supor também que o endereço inicial da sequencia, ou endereço base, esteja em **$s3**. Compile esta instrução de atribuição:
- g = h + A[8];
  - **lw $t0, 32($s3)**
  - **add $s1, $s2, $t0**

Operandos do Hardware do Computador

- Endereçamento Big end:
  - Usa o byte mais a **esquerda** para endereçamento
  - MIPS está nesta categoria
- Endereçamento Little end:
  - Usa o byte mais a **direita** para endereçamento

Operandos do Hardware do Computador

- Instrução Store Word → sw
  - Armazenar os dados de um registrador na RAM
- O formato da instrução **sw**
  - 1º argumento: registrador de onde o dado será copiado
  - 2º argumento: uma constante (**offset**)
  - 3º argumento: registrador usado para armazenar o valor da memória (**registrador base**)

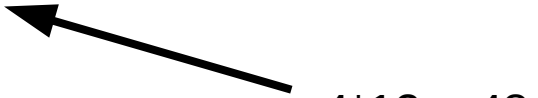
Operandos do Hardware do Computador

- Suponha que a variável h esteja associada ao registrador **$s2**, e o endereço base do array A esteja em **$s3**. Qual é o código assembly do MIPS para a instrução de atribuição a seguir:
- A[12] = h + A[8];

Operandos do Hardware do Computador

$s3 $s2 $s3
A[12] = h + A[8];

- Copiar A[8] para um registrador temporário
  - **lw $t0, 32($s3)**
- Operação aritmética normal
  - **add $t0, $s2, $t0**
- Salvar o valor temporário no array A[12]
  - **sw $t0, 48($s3)**



Operandos do Hardware do Computador

- Em geral os programas possuem mais variáveis que registradores
  - Instruções são realizadas utilizando registradores
- Compilador tenta manter as variáveis mais utilizadas nos registradores
- Processo de colocar variáveis menos utilizadas na memória:
  - **Spilling registers**

Operandos do Hardware do Computador

- Mais da metade das operações aritméticas são feitas usando constantes
  - Ex: x = x + 1;
- Os processadores tem instruções especiais para estes eventos

Operandos do Hardware do Computador

- Vamos supor que queremos adicionar um valor constante **4** ao registrador **$s3**.
- Primeiro temos que carregá-lo em um registrador
- Depois fazemos a operação aritmética

```
lw  $t0, EndConstante4($s1)  # $t0 = constante 4
add $s3,$s3,$t0              # $s3 = $s3 + $t0 ($t0 == 4)
```

Operandos do Hardware do Computador

- Existe instruções aritméticas quanto um dos operando é uma constante
  - Evita uma instrução load word
- Add imediato → **addi**
- O formato da instrução **addi**
  - 1º argumento: registrador destino
  - 2º argumento: registrador usado para a soma
  - 3º argumento: valor constante

Operandos do Hardware do Computador

- Vamos supor que queremos adicionar um valor constante **4** ao registrador **$s3**.
  - **addi $s3, $s3, 4** **$s3 = $s3 + 4**

Operandos do Hardware do Computador

- Em desempenho vimos que uma das maneiras para otimizar um programa é:
  - Otimizar os casos mais comuns
- Operandos constantes aparecem mais da metade das vezes nas operações aritméticas

- **Princípio de projeto 03:**
  - Agilize os casos mais comuns

Operandos do Hardware do Computador

Operandos MIPS

Nome	Exemplo	Comentários
32 registradores	`$s0, $s1, ..., $t0, $t1, ...`	Locais rápidos para dados. No MIPS, os dados precisam estar em registradores para a realização de operações aritméticas.
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 seqüenciais diferem em 4 vezes. A memória contém estruturas de dados, arrays e spilled registers.

Linguagem assembly do MIPS

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Três operandos; dados nos registradores
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2- $s3`	Três operandos; dados nos registradores
	add immediate	`addi $s1,$s2,100`	`$s1 = $s2 + 100`	Usada para somar constantes
Transferência de dados	load word	`lw $s1,100($s2)`	`$s1 = Memória[$s2 + 100]`	Dados da memória para o registrador
	store word	`sw $s1,100($s2)`	`Memória[$s2 + 100] = $s1`	Dados do registrador para a memória

Representando Instruções no Computador

- As instruções são mantidas como uma séria de dígitos binários
  - Cada parte da instruções pode ser considerada um número individual
  - A instrução completa é a colocação de cada sequencia binária lado a lado

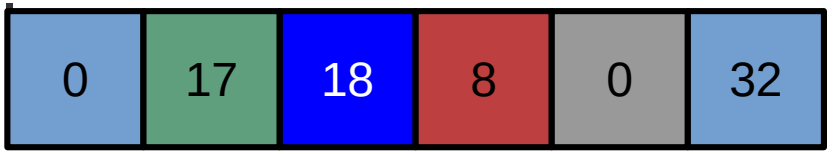
Representando Instruções no Computador

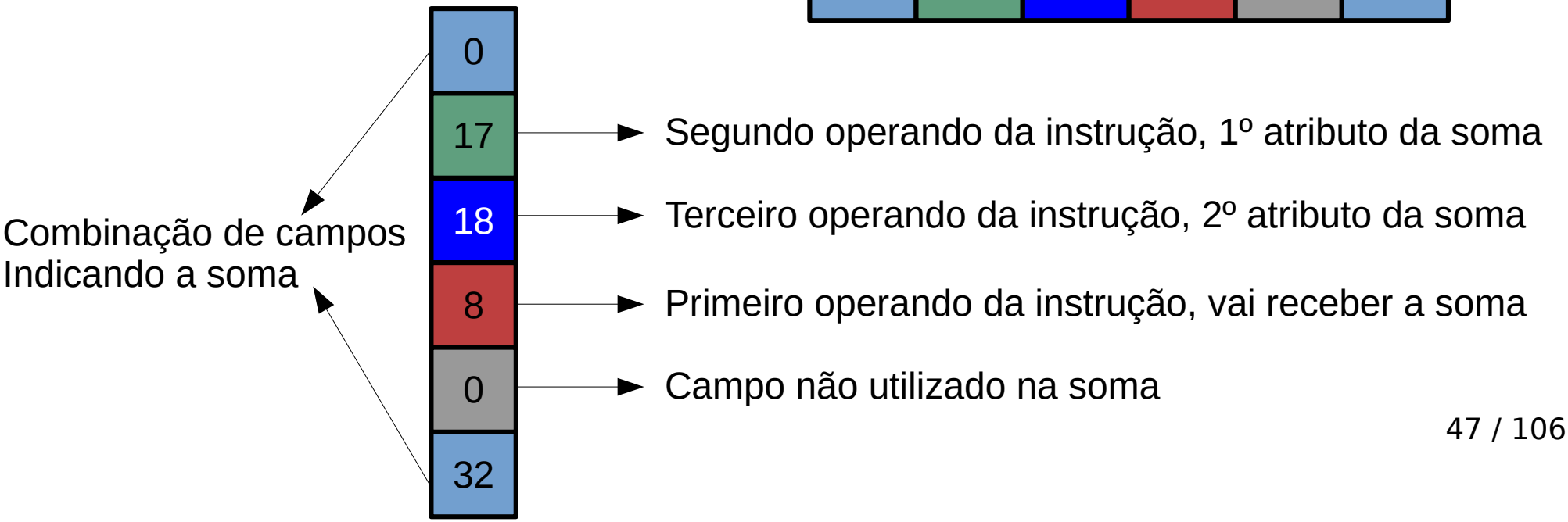
- Em assembly MIPS
  - 32 registradores
  - $s0 a $s7 → registradores de 16 a 23
  - $t0 a $t7 → registradores de 8 a 15

	Name	Register number	Usage
$0	$zero	0	the constant value 0
.	$v0-$v1	2-3	values for results and expression evaluation
.	$a0-$a3	4-7	arguments
.	$t0-$t7	8-15	temporaries
	$s0-$s7	16-23	saved
	$t8-$t9	24-25	more temporaries
	$gp	28	global pointer
	$sp	29	stack pointer
	$fp	30	frame pointer
$31	$ra	31	return address

Representando Instruções no Computador

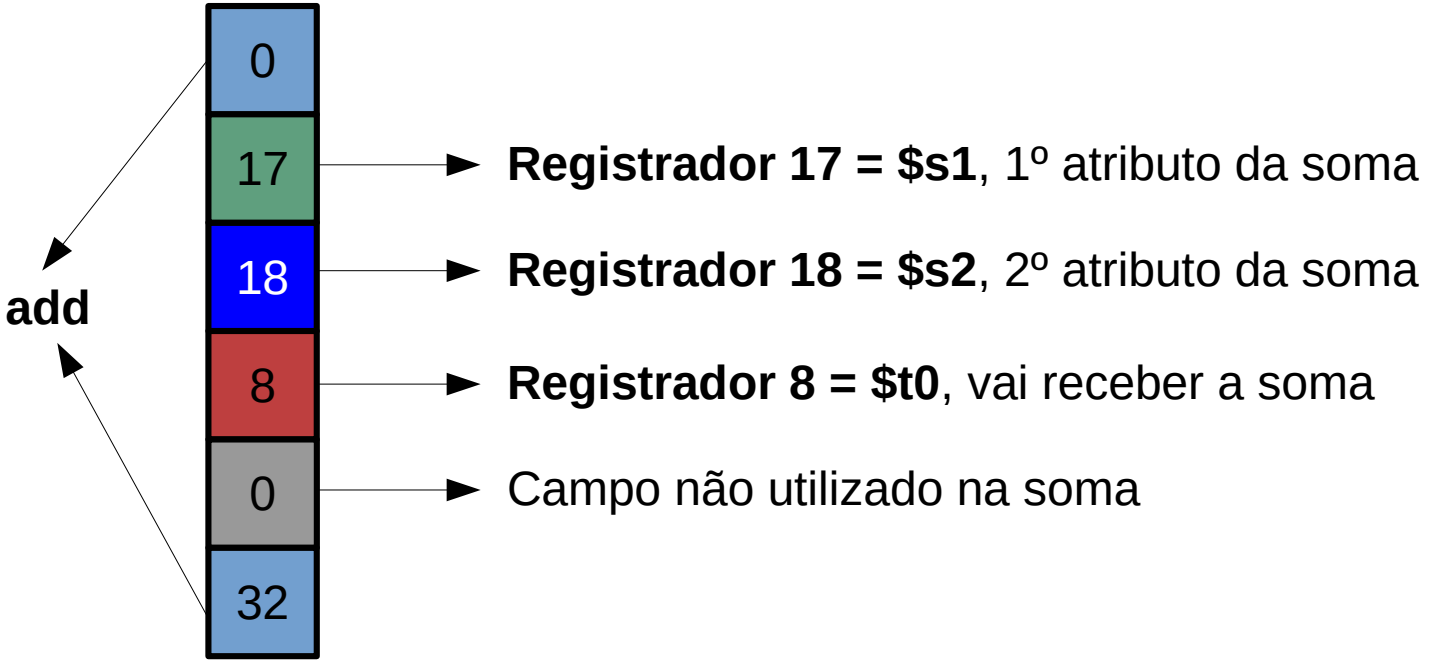
- **Traduzindo uma instrução MIPS para linguagem de máquina**
- Comando em assembly MIPS:
  - **add $t0, $s1, $s2**
- Representação decimal:





Representando Instruções no Computador

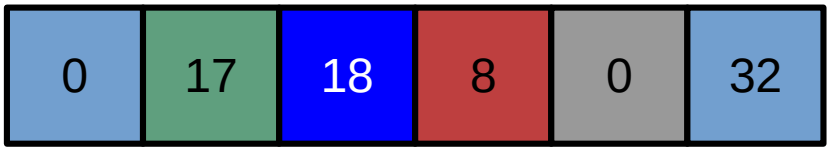
- Representação decimal:



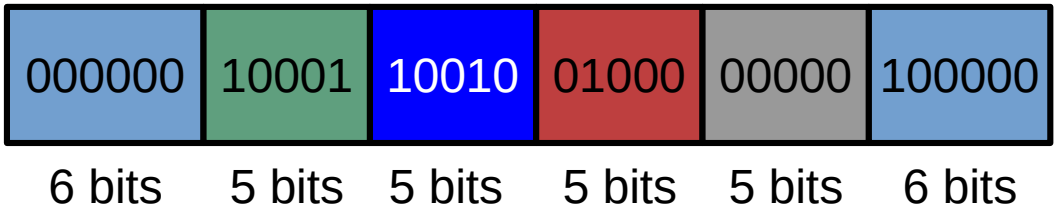
- $s0 a $s7 → registradores de 16 a 23
- $t0 a $t7 → registradores de 8 a 15

Representando Instruções no Computador

- Representação decimal:



- Representação binária:



- **Toda instrução de MIPS possui 32 bits de largura**
  - Mesmo tamanho que a nossa palavra de dados (word)
  - **Princípio de projeto 01**:
    - Simplicidade favorece a regularidade

Representando Instruções no Computador

- Versão numérica das instruções
  - **Linguagem de máquina**
- Sequencia de instruções
  - **Código de máquina**
  - Ex: compiladores geram código de máquina

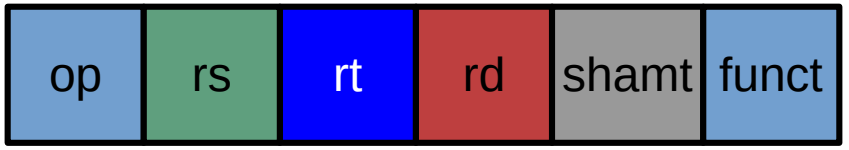
Representando Instruções no Computador

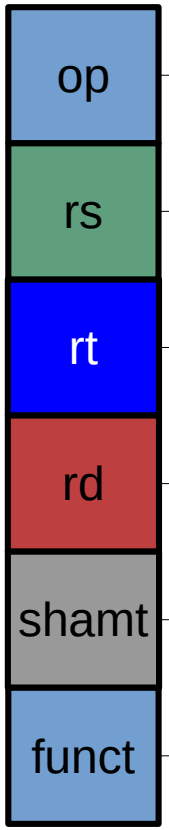
- Uma alternativa para não representar os número em binário é utilizar a base hexadecimal
  - Base 16 é potencia de 2, logo são compatíveis
  - Substitui cada grupo de 4 dígitos por um em hexadecimal

Binário	1010	0110	0000	1100
Hexadecimal	A	6	0	C
Valor Final	A60C (Hexadecimal)			

Campos do MIPS

- Os campos do MIPS recebem nomes:





Opcode, operação básica da instrução

Registrador do primeiro operando de origem (“**s**ource **r**egister”)

Registrador do segundo operando de origem (letra “**t**” segue a letra “**s**”)

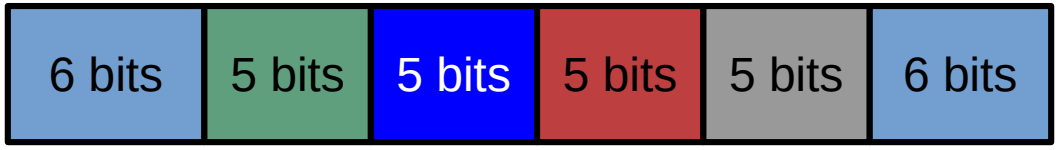
Registrador do operando de **destino**, recebe resultado da operação

Shitf amount, quantidade de deslocamento

Função, seleciona a variante da operação **Opcode**
Chamado também de **código da função**

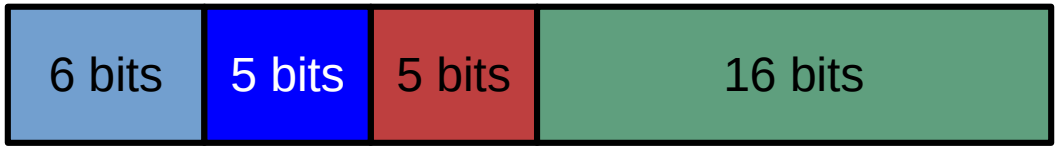
Campos do MIPS

- Problema:
  - Load Word
    - **lw $registrador_destino, offset($registrador_base)**
    - O offset terá apenas 5 bits para representar o deslocamento na memória
      - $2^5 = 32$



Campos do MIPS

- Solução:
  - Utilizar outra estrutura para a instrução **lw**
    - Em vez de direcionar 5 bits para o deslocamento, iremos direcionar 16 bits
      - $2^{16} = 65.536$
      - Equivalente ao inteiro de arquiteturas 32 bits

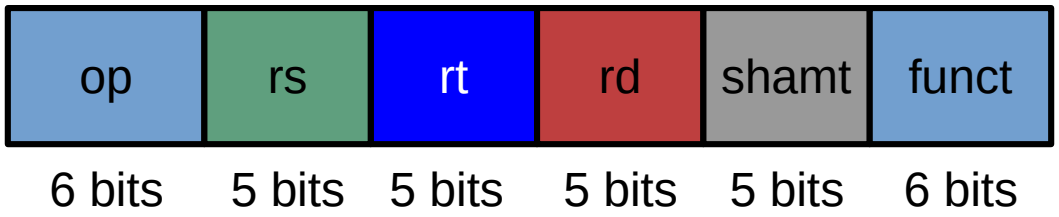


Campos do MIPS

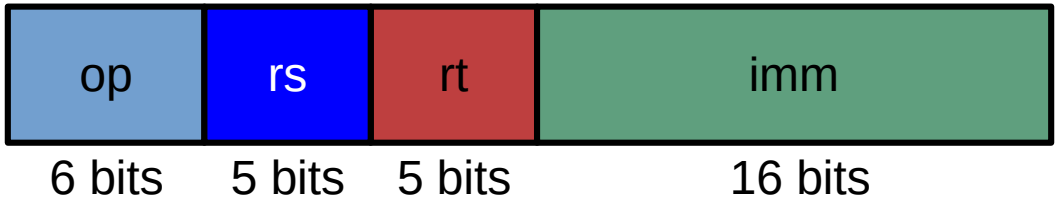
- **Princípio de projeto 04:**
  - Um bom projeto exige bons compromissos
- Todas as instruções devem ter o mesmo tamanho
  - 32 bits
- Podemos ter diferentes estruturas para as instruções
  - Sempre respeitando o tamanho

Campos do MIPS

- Formatos de instrução:
  - **Tipo-R (de registrador)**



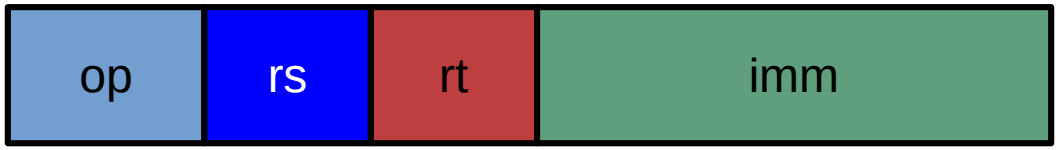
  - **Tipo-I (de imediato “imm**ediate**”)**



Se houvessem mais de 32 registradores como ficaria o endereçamento?

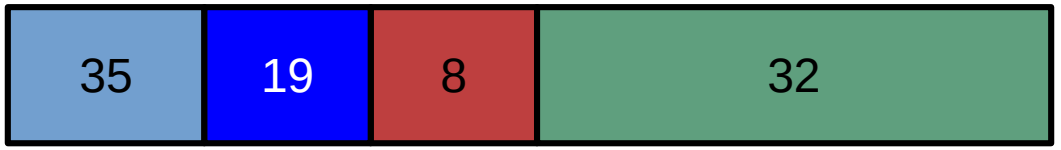
Campos do MIPS

- Exemplo:
  - **lw $t0, 32($s3)** # Registrador $t0 recebe A[8]



$t0 = registrador 8 (destino dos dados)

- **$s3** = registrador 19 (origem dos dados)
- **32** = constante imediata
- **lw** = operação 35



Campos do MIPS

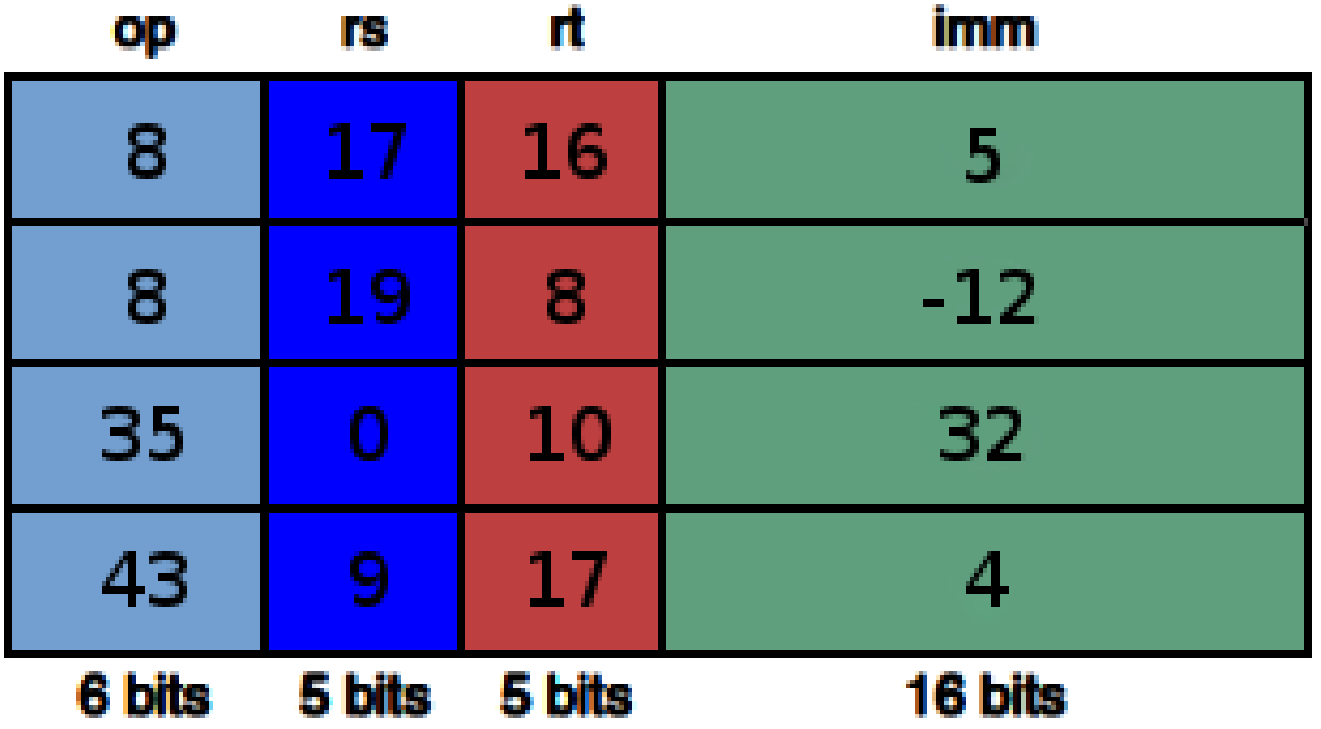
- Exemplo:

```
addi $s0, $s1, 5
addi $t0, $s3, -12
lw   $t2, 32($0)
sw   $s1,  4($t1)
```

Destino

Origem e Deslocamento

Source **R**egister



Campos do MIPS

- Resumo:

Instrução	Formato	op	rs	rt	rd	Shamt	Funct	endereço
add	R	0	reg	reg	reg	0	32_{dec}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{dec}	n.a.
add immediate	I	8_{dec}	reg	reg	n.a.	n.a.	n.a.	constante
lw (load word)	I	35_{dec}	reg	reg	n.a.	n.a.	n.a.	endereço
sw (store word)	I	43_{dec}	reg	reg	n.a.	n.a.	n.a.	endereço

Linguagem assembly do MIPS

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Três operandos; dados nos registradores
	subtract	sub $s1,$s2,$s3	$s1 = $s2- $s3	Três operandos; dados nos registradores
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	Usada para somar constantes
Transferência de dados	load word	lw $s1,100($s2)	$s1 = Memória[$s2 + 100]	Dados da memória para o registrador
	store word	sw $s1,100($s2)	Memória[$s2 + 100] = $s1	Dados do registrador para a memória

Campos do MIPS

- Exercício:

Agora, já podemos usar um exemplo completo daquilo que o programador escreve até o que o computador executa. Se `$t1` possui a base do array `A` e `$s2` corresponde a `h`, então a instrução de atribuição

```
A[300] = h + A[300];
```

é compilada para

```
lw   $t0,1200($t1)  # Reg. temporário $t0 recebe A[300]
add  $t0,$s2,$t0    # Reg. temporário $t0 recebe h + A[300]
sw   $t0,1200($t1)  # Armazena h + A[300] de volta para A[300]
```

Qual o código em linguagem de máquina MIPS para essas três instruções?

Campos do MIPS

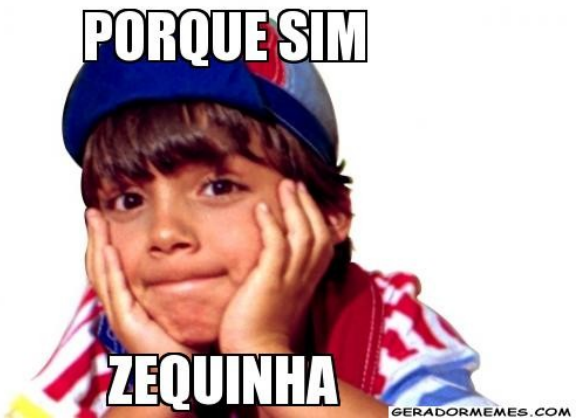
- Resposta:
  - **lw $t0, 1200($t1)**
  - **add $t0, $s2, $t0**
  - **sw $t0, 1200($t1)**

op	rs	rt	rd	endereço/shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Representando Instruções no Computador

- Por que não existe a Subtração Imediata (**subi**)?
  - Valores negativos aparecem menos que positivos
  - O campo imediato **IMM** pode enviar valores positivos e negativos
- Então por que existe a instrução **sub**?



Operações Lógicas

- Qual o valor final das operações em **Java** ou **C**?

int x = 2; x = x **<<** 1;	int y = 3; y = y **>>** 1;	int z = 5 **&** 2;	int w= 5**|**2;

Operações Lógicas

- Qual o valor final das operações em **Java** ou **C**?

int x = 2; x = x **<<** 1;	int y = 3; y = y **>>** 1;	int z = 5 **&** 2;	int w= 5**|**2;
4	**1**	**0**	**7**

Operações Lógicas

- Operações lógicas e suas correspondências em MIPS

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

Operações Lógicas

- Shift a Esquerda – Shift Left Logical (**sll**)
  - **sll a, b, c** **a = b << c**
- O formato da instrução **sll**
  - 1º argumento: registrador destino
  - 2º argumento: registrador onde será dado o shift
  - 3º argumento: valor constante

srl é similar ao **sll**

Operações Lógicas

- Shift a Esquerda – Shift Left Logical (**sll**)
  - **sll $t2, $s0, 4** **$t2 = $s0 << 4**
    - **$s0**

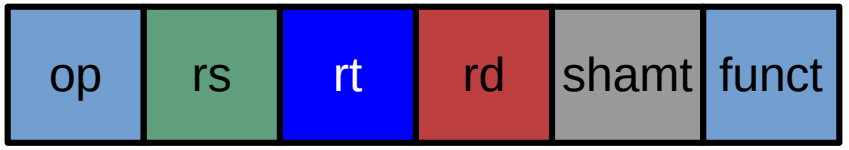
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_{bin} = 9_{dec}$

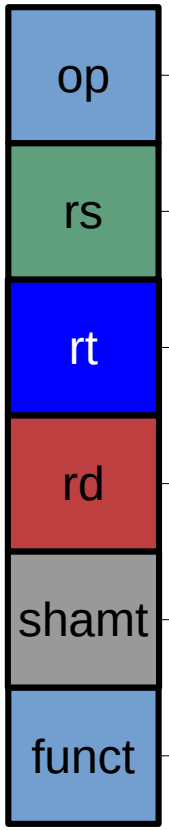
    - **$t2**

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_{bin} = 144_{dec}$

Campos do MIPS

- Os campos do MIPS recebem nomes:





Opcode, operação básica da instrução

Registrador do primeiro operando de origem ("**s**ource **r**egister")

Registrador do segundo operando de origem (letra "**t**" segue a letra "**s**")

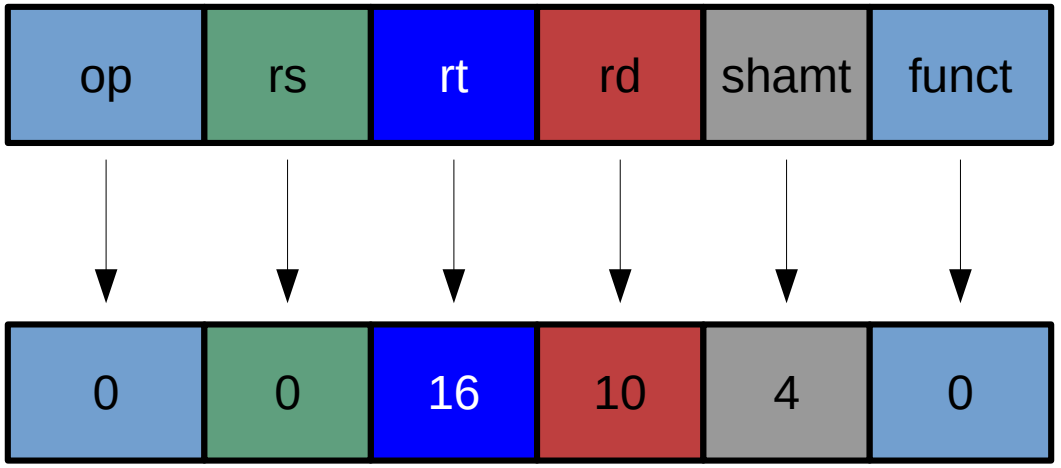
Registrador do operando de **destino**, recebe resultado da operação

Shitf amount, quantidade de deslocamento

Função, seleciona a variante da operação **Opcode**
Chamado também de **código da função**

Operações Lógicas

- Shift a Esquerda – Shift Left Logical (**sll**)
  - **sll $t2, $s0, 4** **$t2 = $s0 << 4**
  - Operação **0** com função **0**
  - Campo **rs** não é utilizado



- OBS: os Shift's são do tipo R
  - Não há necessidade de dar um shift de mais de 31 casas (2^5)

Operações Lógicas

- AND (**and**)
  - **and a, b, c** **a = b & c**
- O formato da instrução **and**
  - 1º argumento: registrador destino
  - 2º argumento: primeiro registrador da operação
  - 3º argumento: segundo registrador da operação

or é similar ao **and**

Operações Lógicas

- AND (**and**)
  - **and $t0, $t1, $t2** **a = b & c**
    - **$t1**

$$0000\ 0000\ 0000\ 0000\ 0011\ 1100\ 0000\ 0000_{bin}$$

$$0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 0000\ 0000_{bin}$$

$$0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 0000\ 0000_{bin}$$

Operações Lógicas

- NOT OR (**nor**)
  - A operação **not** utiliza apenas um operando
    - z = ~ x
  - A operação **nor** utiliza dois operandos
    - z = ~ (x | y)

- **Nossas instruções possuem 3 operandos!**

Operações Lógicas

- NOT OR (**nor**)

TABELA VERDADE - NOR		
A	**B**	**~(A+B)**
0	0	1
0	1	0
1	0	0
1	1	0

Operações Lógicas

- NOT OR (**nor**)
  - **nor a, b, c** **a = ~ (b | c)**
- O formato da instrução **nor**
  - 1º argumento: registrador destino
  - 2º argumento: primeiro registrador da operação
  - 3º argumento: segundo registrador da operação

- Em geral usamos o 3º argumento como **$zero** para fazer o **not**

Operações Lógicas

- NOT OR (**nor**)
  - **nor $t0, $t1, $zero** **$t0 = ~ ($t1 | $zero)**
    - **$t1**

      0000 0000 0000 0000 0011 1100 0000 0000$_{bin}$
    - **$t0**

      1111 1111 1111 1111 1100 0011 1111 1111$_{bin}$

Operações Lógicas

- Resumo

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Três operandos
	subtract	sub $s1,$s2,$s3	$s1 = $s2- $s3	Três operandos
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constante
Lógica	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Três operadores em registrador; AND bit a bit
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Três operadores em registrador; OR bit a bit
	nor	nor $s1,$s2,$s3	$s1 = ~($s2 | $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 | 100	OR bit a bit entre registrador com constante
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Deslocamento à esquerda por constante
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Deslocamento à direita por constante
Transferência de dados	load word	lw $s1,100($s2)	$s1 = Memória[$s2 + 100]	Word da memória para o registrador
	store word	sw $s1,100($s2)	Memória[$s2 + 100] = $s1	Word do registrador para a memória

- OBS: os Shift's são do tipo R
  - Não há necessidade de dar um shift de mais de 31 casas (2^5)

Instruções para tomadas de decisão

- Uma das características do computadores é a capacidade de tomar decisões (If – Else)
- Em MIPS existe as avaliações e são implementadas usando um **GO TO** (vá para)

    se x == 0 → vá para linha 30

    senão → vá para linha 70

Instruções para tomadas de decisão

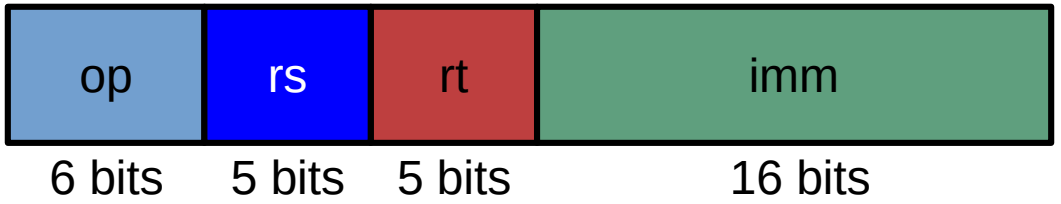
- Desviar se for igual – Branch if equal (**beq**)
  - **beq a, b, c** **if (a==b) → c**
- O formato da instrução **beq**
  - 1º argumento: primeiro registrador da operação
  - 2º argumento: segundo registrador da operação
  - 3º argumento: **rótulo** de uma linha de instrução

Instruções para tomadas de decisão

- Desviar se **não** for igual – Branch if not equal (**bne**)
  - **bne a, b, c** **if (a!=b) → c**
- O formato da instrução **bne**
  - 1º argumento: primeiro registrador da operação
  - 2º argumento: segundo registrador da operação
  - 3º argumento: **rótulo** de uma linha de instrução

Instruções para tomadas de decisão

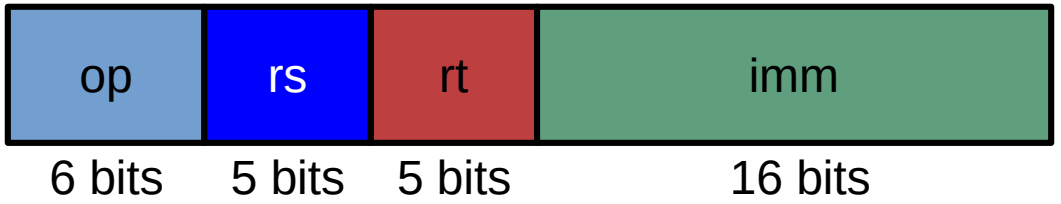
- O formato das instruções de desvio



- 16 bits para o endereço do rotulo é suficiente?
  - Sabendo que o rotulo é um endereço de memória

Instruções para tomadas de decisão

- O formato das instruções de desvio



- 16 bits para o endereço do rotulo é suficiente?
  - **Não!**

Instruções para tomadas de decisão

- ***Solução***: Usar o **c**ontador do **p**rograma (**PC**)
  - **PC** → guarda o endereço da próxima instrução a ser buscada na memória
  - **Endereço de desvio** → informa quantas posições temos que andar para chegar na instrução desejada
    - Valores Negativos ou Positivos
    - 1 bit para o sinal(+,-) e 15 bits para o valor

PC = PC + Endereço de Desvio

Instruções para tomadas de decisão

- Pula para – Jump (**j**)
  - **j a** **goto a**
- O formato da instrução **j**
  - 1º argumento: **rótulo** de uma linha de instrução

- Desvio incondicional → desvio sem avaliação

Instruções para tomadas de decisão

- Formato da instrução jump

Os 26 bits endereçam a faixa de endereços entre os 4 bits mais significativos e os 2 bits menos significativos

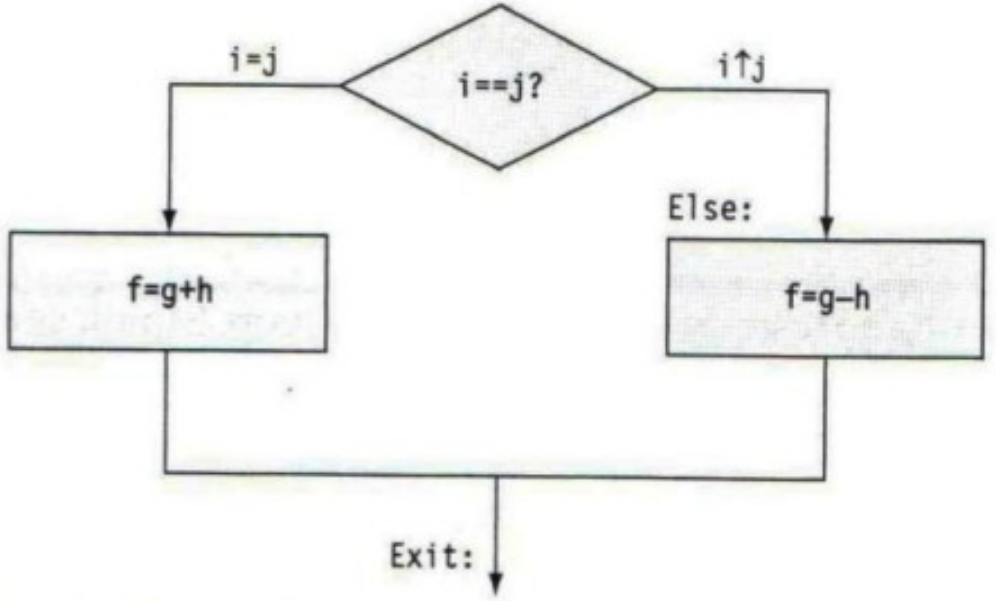
Instruções para tomadas de decisão

- No código abaixo as variáveis f, g, h, i e j estão alocadas nos registradores de $s0 a $s4. Qual o código MIPS para esta instrução em C?

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

Instruções para tomadas de decisão

```
if ( $s3 == $s3 )
    $s0 = $s1 + $s2;
else
    $s0 = $s1 – $s2;
```




```
        bne $s3, $s4, desvio
        add $s0, $s1, $s2
        j saida
desvio: sub $s0, $s1, $s2
saida:
```

Instruções para tomadas de decisão

- Os **montadores** que calculam os endereços dos:
  - rótolos criados pelo programador ou compilador
  - dados nos Loads e Stores

Instruções para tomadas de decisão

- Atribuir se menor que – Set on less than (**slt**)
  - **slt a, b, c** **if (b<c) → a=1; else a=0;**
- O formato da instrução **bne**
  - 1º argumento: registrador de destino
  - 2º argumento: primeiro registrador da comparação
  - 3º argumento: segundo registrador da comparação

Instruções para tomadas de decisão

- Atribuir se menor que **Imediado** – Set on less than i (**slti**)
  - **slti a, b, c** **if (b<c) → a=1; else a=0;**
- O formato da instrução **bnei**
  - 1º argumento: registrador de destino
  - 2º argumento: primeiro registrador da comparação
  - 3º argumento: valor constante da comparação

Instruções para tomadas de decisão

- Exercício

A linguagem C possui muitas instruções para decisões e loops, enquanto o MIPS possui poucas. Quais dos seguintes itens explicam ou não explicam esse desequilíbrio? Por quê?

1. Mais instruções de decisão tornam o código mais fácil de ler e entender.
2. Menos instruções de decisão simplificam a tarefa da camada inferior responsável pela execução.
3. Mais instruções de decisão significam menos linhas de código, o que geralmente reduz o tempo de codificação.
4. Mais instruções de decisão significam menos linhas de código, o que geralmente resulta na execução de menos operações.

Instruções para tomadas de decisão

- Exercício

A linguagem C possui muitas instruções para decisões e loops, enquanto o MIPS possui poucas. Quais dos seguintes itens explicam ou não explicam esse desequilíbrio? Por quê?

1. Mais instruções de decisão tornam o código mais fácil de ler e entender.
2. Menos instruções de decisão simplificam a tarefa da camada inferior responsável pela execução.
3. Mais instruções de decisão significam menos linhas de código, o que geralmente reduz o tempo de codificação.
4. Mais instruções de decisão significam menos linhas de código, o que geralmente resulta na execução de menos operações.

Instruções para tomadas de decisão

- Exercício

Por que a linguagem C oferece dois conjuntos de operadores para AND (& e &&) e dois conjuntos de operadores para OR (| e ||), enquanto o MIPS não faz isso?

1. As operações lógicas AND e OR implementam & e |, enquanto os desvios condicionais implementam && e ||.
2. A afirmativa anterior é o contrário: && e || correspondem a operações lógicas, enquanto & e | são mapeados para desvios condicionais.
3. Elas são redundantes e significam a mesma coisa: && e || são simplesmente herdados da linguagem de programação B, a antecessora do C.

Instruções para tomadas de decisão

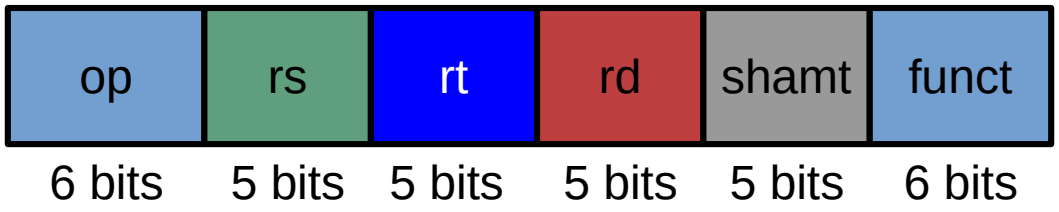
- Exercício

Por que a linguagem C oferece dois conjuntos de operadores para AND (& e &&) e dois conjuntos de operadores para OR (| e ||), enquanto o MIPS não faz isso?

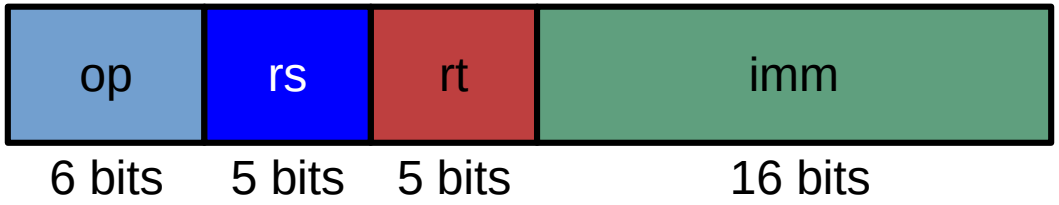
1. As operações lógicas AND e OR implementam & e |, enquanto os desvios condicionais implementam && e ||.
2. A afirmativa anterior é o contrário: && e || correspondem a operações lógicas, enquanto & e | são mapeados para desvios condicionais.
3. Elas são redundantes e significam a mesma coisa: && e || são simplesmente herdados da linguagem de programação B, a antecessora do C.

Formatos das Instruções (R, I e J)

1)R



2)I



3)J

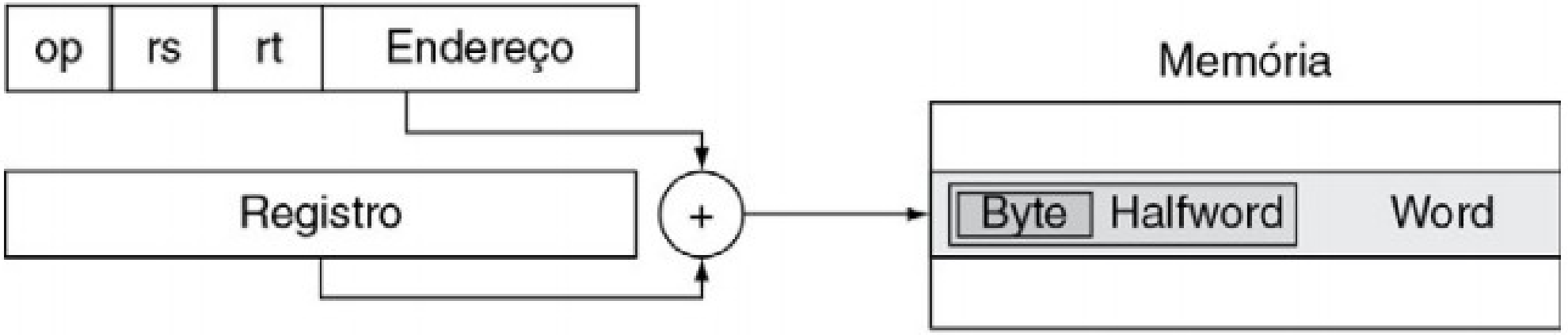
Nome	Campos						Comentários
Tamanho do campo	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções do MIPS têm 32 bits
Formato R	op	rs	rt	rd	shamt	funct	Formato das instruções aritméticas
Formato I	op	rs	rt	endereço/imediato			Formato imediato para transferências e desvios
Formato J	op	endereço de destino					Formato das instruções de jump

Modos de Endereçamento

1) Endereçamento em **registrador**:
   - Operandos são registradores
   - Ex: add, sub, ...

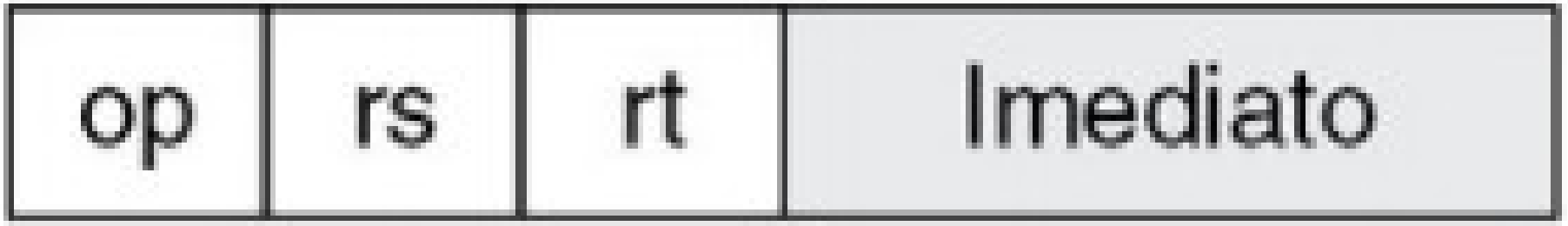
Modos de Endereçamento

2) Endereçamento de **base ou deslocamento**:
   - Operando está num local de memória cujo endereço é a soma de uma constante com um registrador
   - Ex: lw, sw, sll, srl



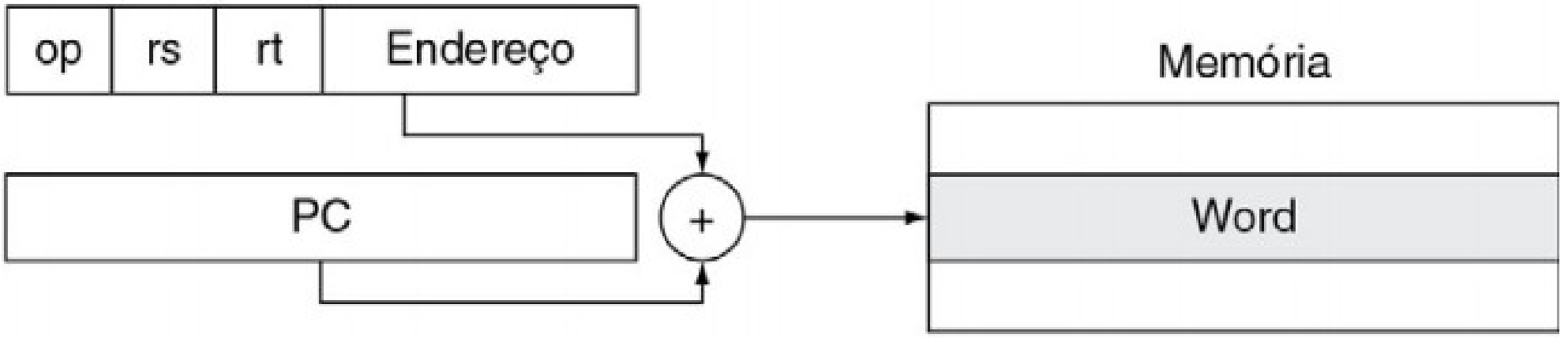
Modos de Endereçamento

3) Endereçamento **imediato**:
   - Operando é uma contante dentro da instrução
   - Ex: addi, slti, ori



Modos de Endereçamento

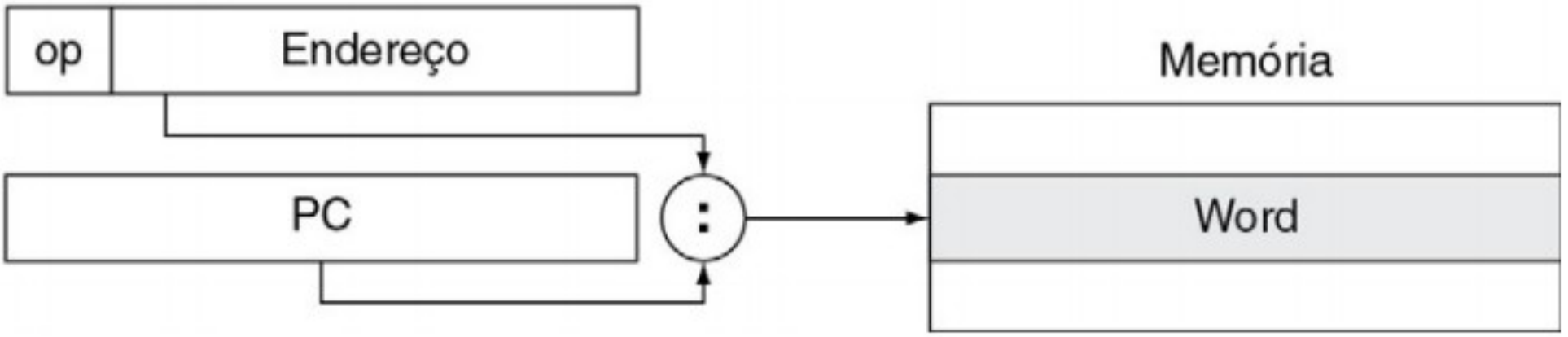
4) Endereçamento **relativo**:
   - Endereço é a soma do **PC** com uma constante
   - Ex: desvios (beq, bne)



Modos de Endereçamento

5) Endereçamento **pseudodireto**

- Endereço de jump são os 26 bits da instrução concatenados com os bits mais altos do **PC**
- Ex: jump



Resumo das instruções

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	Add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Três operandos; dados nos registradores
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2- $s3`	Três operandos; dados nos registradores
Transferência de dados	load word	`lw $s1,100($s2)`	`$s1 = Memória[$s2 + 100]`	Dados da memória para o registrador
	store word	`sw $s1,100($s2)`	`Memória[$s2 + 100] = $s1`	Dados do registrador para a memória
Lógica	And	`and $s1,$s2,$s3`	`$s1 = $s2 & $s3`	Três operadores em registrador; AND bit a bit
	Or	`or $s1,$s2,$s3`	`$s1 = $s2 | $s3`	Três operadores em registrador; OR bit a bit
	nor	`nor $s1,$s2,$s3`	`$s1 = ~($s2 | $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 | 100`	OR bit a bit entre registrador com constante
	shift left logical	`sll $s1,$s2,10`	`$s1 = $s2 << 10`	Deslocamento à esquerda por constante
	shift right logical	`srl $s1,$s2,10`	`$s1 = $s2 >> 10`	Deslocamento à direita por constante
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	`slt $s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compara menor que imediato; usado com beq, bne
Desvio incondicional	jump	`j L`	go to L	Desvia para endereço de destino

Resumo das instruções

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	Add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Três operandos; dados nos registradores
	subtract	sub $s1,$s2,$s3	$s1 = $s2- $s3	Três operandos; dados nos registradores
Transferência de dados	load word	lw $s1,100($s2)	$s1 = Memória[$s2 + 100]	Dados da memória para o registrador
	store word	sw $s1,100($s2)	Memória[$s2 + 100] = $s1	Dados do registrador para a memória
Lógica	And	and $s1,$s2,$s3	$s1 = $s2 & $s3	Três operadores em registrador; AND bit a bit
	Or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Três operadores em registrador; OR bit a bit
	nor	nor $s1,$s2,$s3	$s1 = ~($s2 | $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 | 100	OR bit a bit entre registrador com constante
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Deslocamento à esquerda por constante
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Deslocamento à direita por constante
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	slt $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compara menor que imediato; usado com beq, bne
Desvio incondicional	jump	j L	go to L	Desvia para endereço de destino

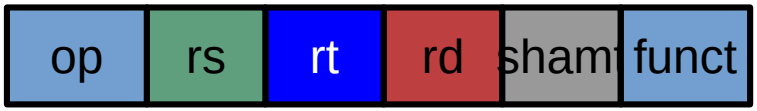
Registrador - Imediato - Jump

Exercício

- Ligue os modos de endereçamento aos seus respectivos formatos de instrução:

1) Endereçamento em **registrador**
2) Endereçamento de **base ou deslocamento**
3) Endereçamento **imediato**:
4) Endereçamento **relativo**:
5) Endereçamento **pseudodireto**

A)R



B)I

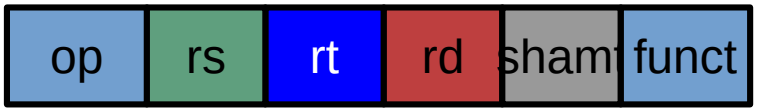
C)J

Exercício

- Ligue os modos de endereçamento do lado direito aos seus respectivos formatos de instrução do lado esquerdo:

1) Endereçamento em **registrador**
2) Endereçamento de **base ou deslocamento**
3) Endereçamento **imediato**:
4) Endereçamento **relativo**:
5) Endereçamento **pseudodireto**

A)R



B)I

C)J

Exercício

- Ligue as instruções aos seus respectivos formatos de instrução:

1) `sll $t1, $s3, 2`

2) `j loop`

3) `lw $t1, 12($s6)`

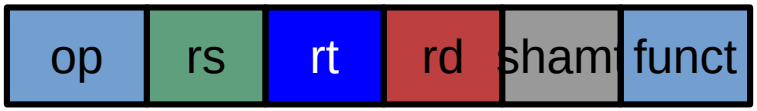
4) `bne $t1, $s5, saida`

5) `addi $s3, $s3, 1`

6) `and $s3, $s3, $t0`

7) `ori $s2, $s0, 3`

A) R



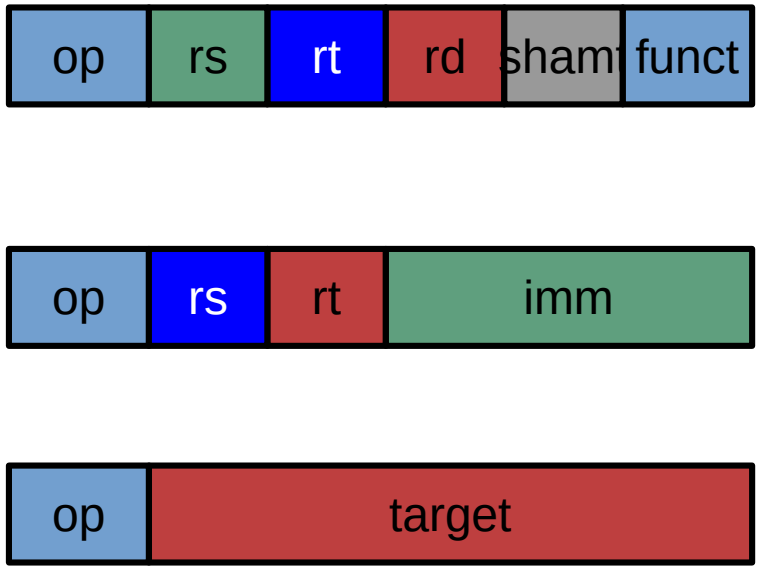
B) I

C) J

Exercício

- Ligue as instruções aos seus respectivos formatos de instrução:

1) sll $t1, $s3, 2
2) j loop
3) lw $t1, 12($s6)
4) bne $t1, $s5, saida
5) addi $s3, $s3, 1
6) and $s3, $s3, $t0
7) ori $s2, $s0, 3



Arquitetura de Computadores

Unichristus
Centro Universitário Christus

Unidade 03: Instruções: a Linguagem de Máquina

Daniel Teixeira

prof.danielnt@gmail.com
dant25.github.io/professor