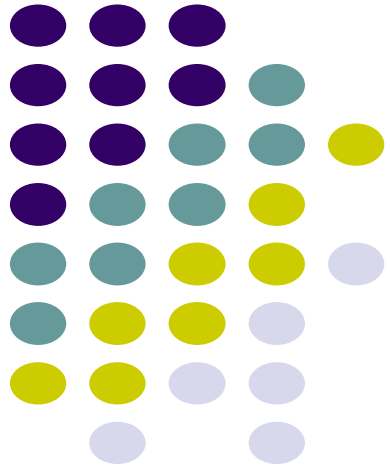
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Arquitetura e Organização de Computadores

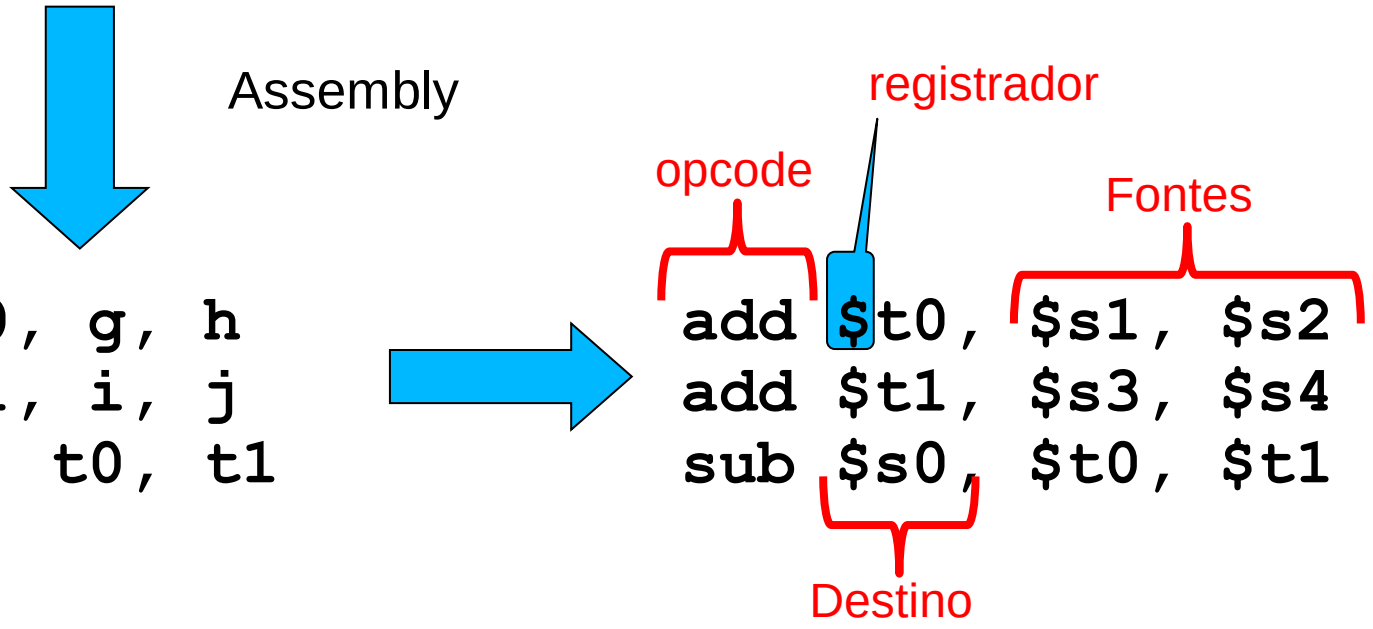
Instruções: a linguagem do computador
Parte II



Prof. Sílvio Fernandes

Na aula passada...

- Um programa em alto nível
  - f = (g + h) – (i + j);



Na aula passada...

- Instruções aritméticas

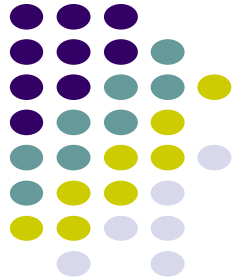
Categoria	Nome	Exemplo	Operação	Comentários
Aritmética	add	add $8, $9, $10	$8 = $9 + $10	*Overflow* gera exceção
	sub	sub $8, $9, $10	$8 = $9 – $10	*Overflow* gera exceção
	addi	addi $8, $9, 40	$8 = $9 + 40	*Overflow* gera exceção Valor do imediato na faixa entre –32.768 e +32.767
	addu	addu $8, $9, $10	$8 = $9 + $10	*Overflow* não gera exceção
	subu	subu $8, $9, $10	$8 = $9 – $10	*Overflow* não gera exceção
	addiu	addiu $8, $9, 40	$8 = $9 + 40	*Overflow* não gera exceção Valor do imediato na faixa entre 0 e 65.535
	mul	mul $8, $9, $10	$8 = $9 x $10	*Overflow* não gera exceção HI, LO imprevisíveis após a operação
	mult	mult $9, $10	HI,LO = $9 x $10	*Overflow* não gera exceção
	multu	multu $9, $10	HI,LO = $9 x $10	*Overflow* não gera exceção
	div	div $9, $10	HI = $9 mod $10 LO = $9 div $10	*Overflow* não gera exceção
	divu	divu $9, $10	HI = $9 mod $10 LO = $9 div $10	*Overflow* não gera exceção

Na aula passada...

- Instruções lógicas

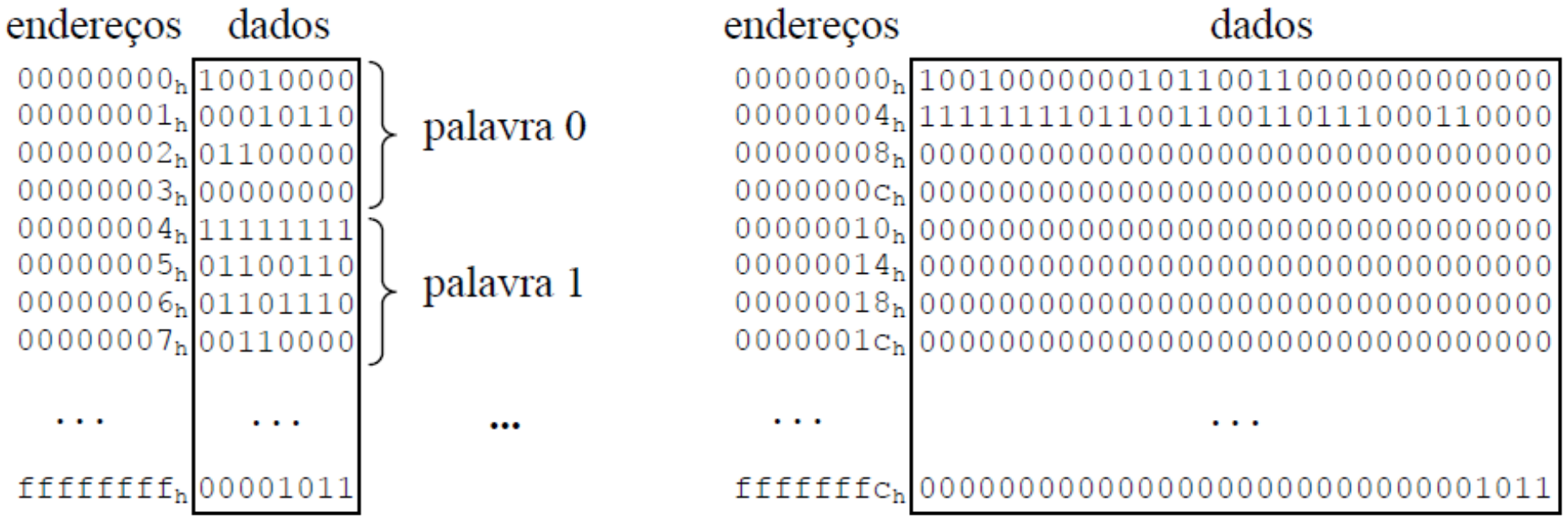
Categoria	Nome	Exemplo	Operação	Comentários
lógicas	or	or $8, $9, $10	$8 = $9 or $10	
	and	and $8, $9, $10	$8 = $9 and $10	
	xor	xor $8, $9, 40	$8 = $9 xor 40	
	nor	nor $8, $9, $10	$8 = $9 nor $10	
	andi	andi $8, $9, 5	$8 = $9 and 5	Imediato em 16 bits
	ori	ori $8, $9, 40	$8 = $9 or 40	Imediato em 16 bits
	sll	sll $8, $9, 10	$8 = $9 << 10	Desloc. ≤ 32
	srl	srl $8, $9, 5	$8 = $9 >> 5	Desloc. ≤ 32
	sra	srl $8, $9, 5	$8 = $9 >> 5	Desloc. ≤ 32, preserva sinal

Registradores no MIPS



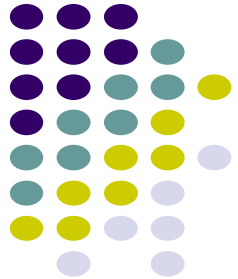
- O processador MIPS possui apenas **32 registradores**. Como fazemos para trabalhar com variáveis complexas cuja quantidade de registradores do MIPS é insuficiente para armazená-las?
  - No MIPS são disponíveis apenas **18 para variáveis**
  - Reposta:
    - Memória!
    - Temos que entender como a memória funciona

Modelo de Memória



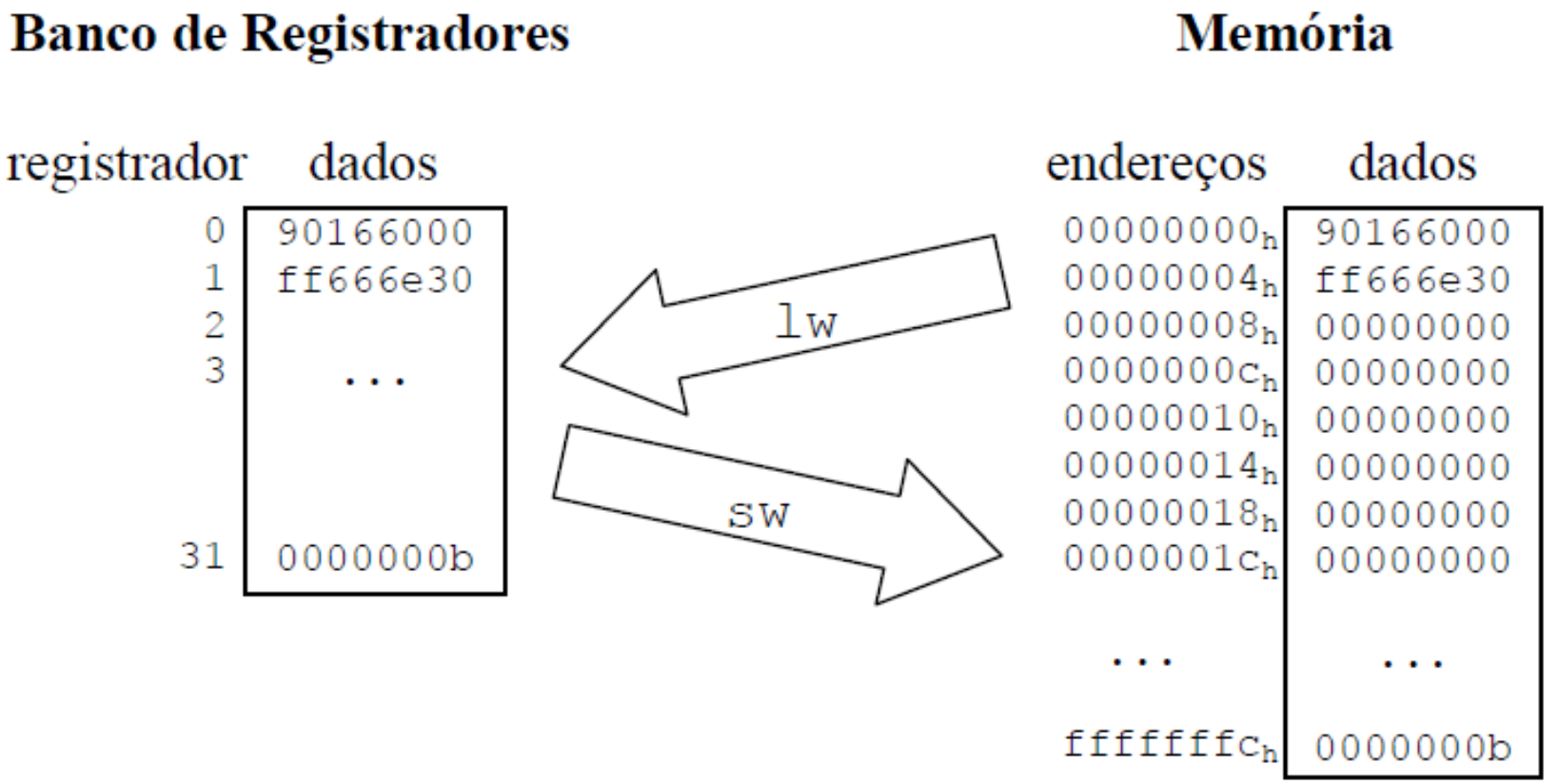
Fonte: WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005

Instruções para transferência de dados



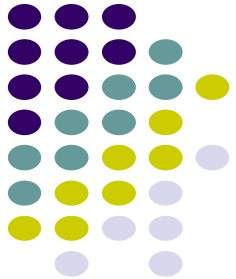
- O MIPS oferece 2 instruções para transferência de **palavras**
  - LOAD WORD (LW)
    - Carrega palavra da memória para um registrador interno
  - STORE WORD (SW)
    - Armazena palavra de um registrador na memória

Instruções para transferência de dados



Fonte: WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005

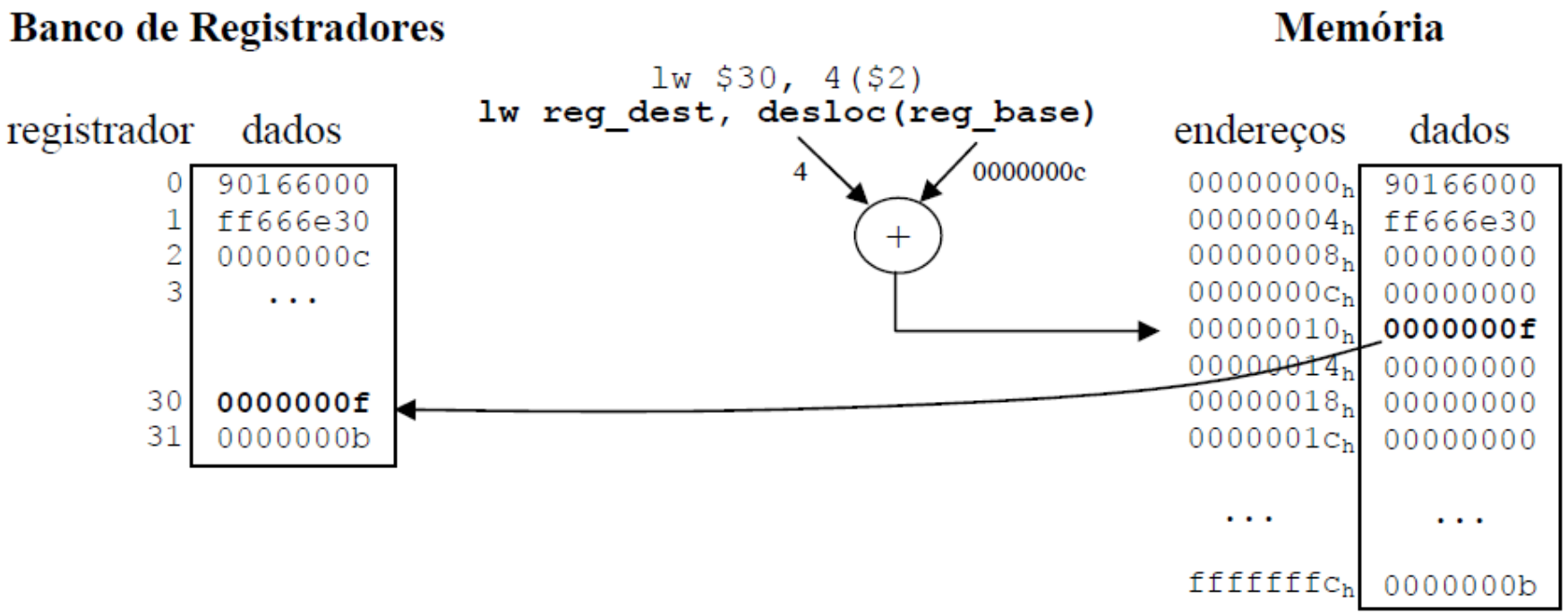
Instruções para transferência de dados



- Utilizando a memória principal.
- Para realizarmos a transferência de um word da memória para um registrador, utilizamos a instrução *lw (load word)*.
  - Sintaxe:
    - **lw $r1, const($r2)**
    - $r1 é o registrador que armazena o conteúdo da memória.
    - const representa um valor constante que é somado ao endereço presente no registrador $r2.
    - $r1 e $r2 são representações abstratas e devem ser substituídos por registradores reais

Instruções para transferência de dados

- Instrução LW



Fonte: WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005

Instruções para transferência de dados

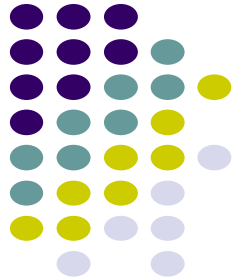
- Qual código assembly para o seguinte trecho de código, considere **A** um array de **inteiros** com o endereço base armazenado em $s3 e a variável **h** em $s2 e **g** em $s1?

```
g = h + A[8];
lw $t0,32($s3);
add $s1,$s2,$t0;
```

$s3	$s2	$s1
A	h	g

  - Porque 32?
    - Porque cada posição do array ocupa 1 palavra (4 bytes ou 32 bits). Lembrem-se que cada endereço sinaliza para uma célula de 1 byte apenas. Logo, para acessarmos o inteiro na posição 8, temos de pular os 8 inteiros que aparecem antes no array, assim temos: 4 * 8 = 32

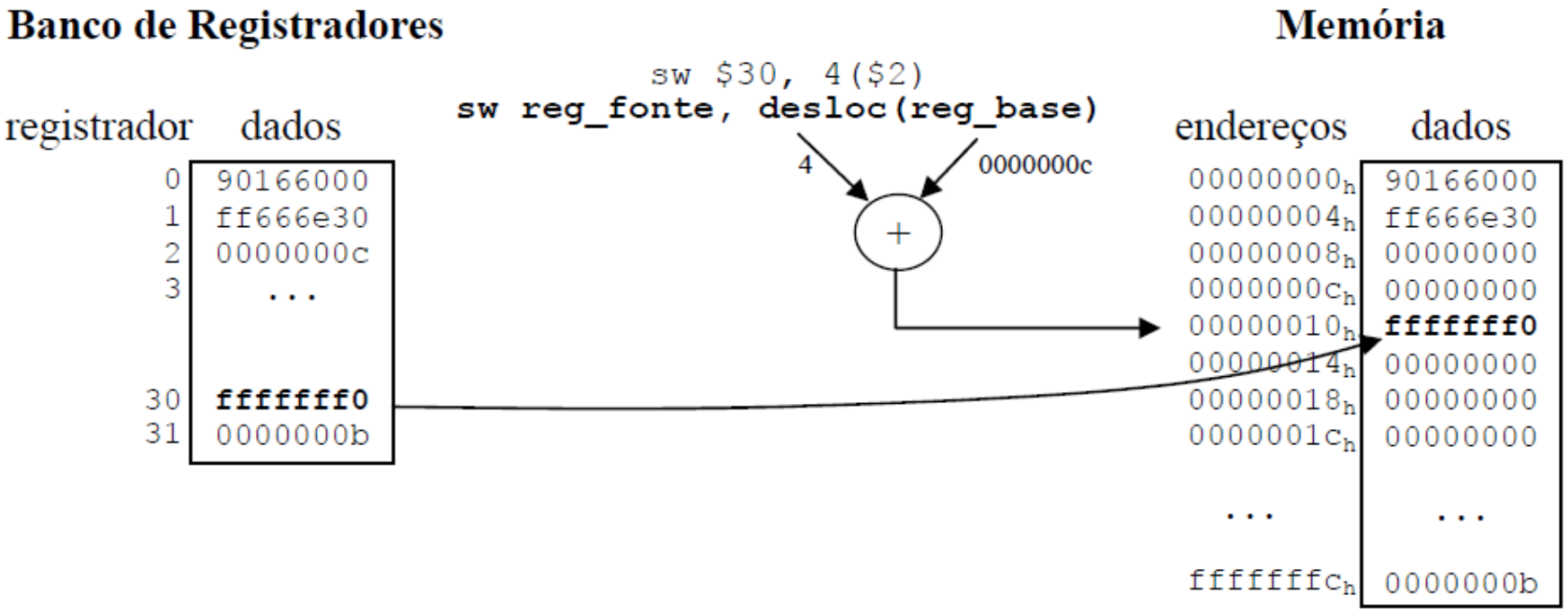
Instruções para transferência de dados



- A instrução que desempenha função inversa ao load word (lw) é a instrução **store word (sw)**.
- Basicamente ela transfere o conteúdo de um registrador para um endereço específico da memória principal.
- Sintaxe:
  - Similar ao do lw.
    - **sw $r1, const($r2)**
    - $r1 contém o valor a ser armazenado no endereço indicado em $r2

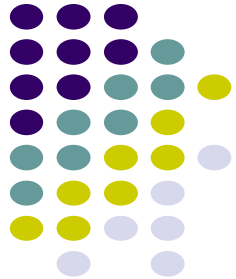
Instruções para transferência de dados

- Instrução SW



Fonte: WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005

Instruções para transferência de dados



- Exemplo:
  - A[12] = h + A[8];
  - Considerando que h está em $s2 e A em $s3

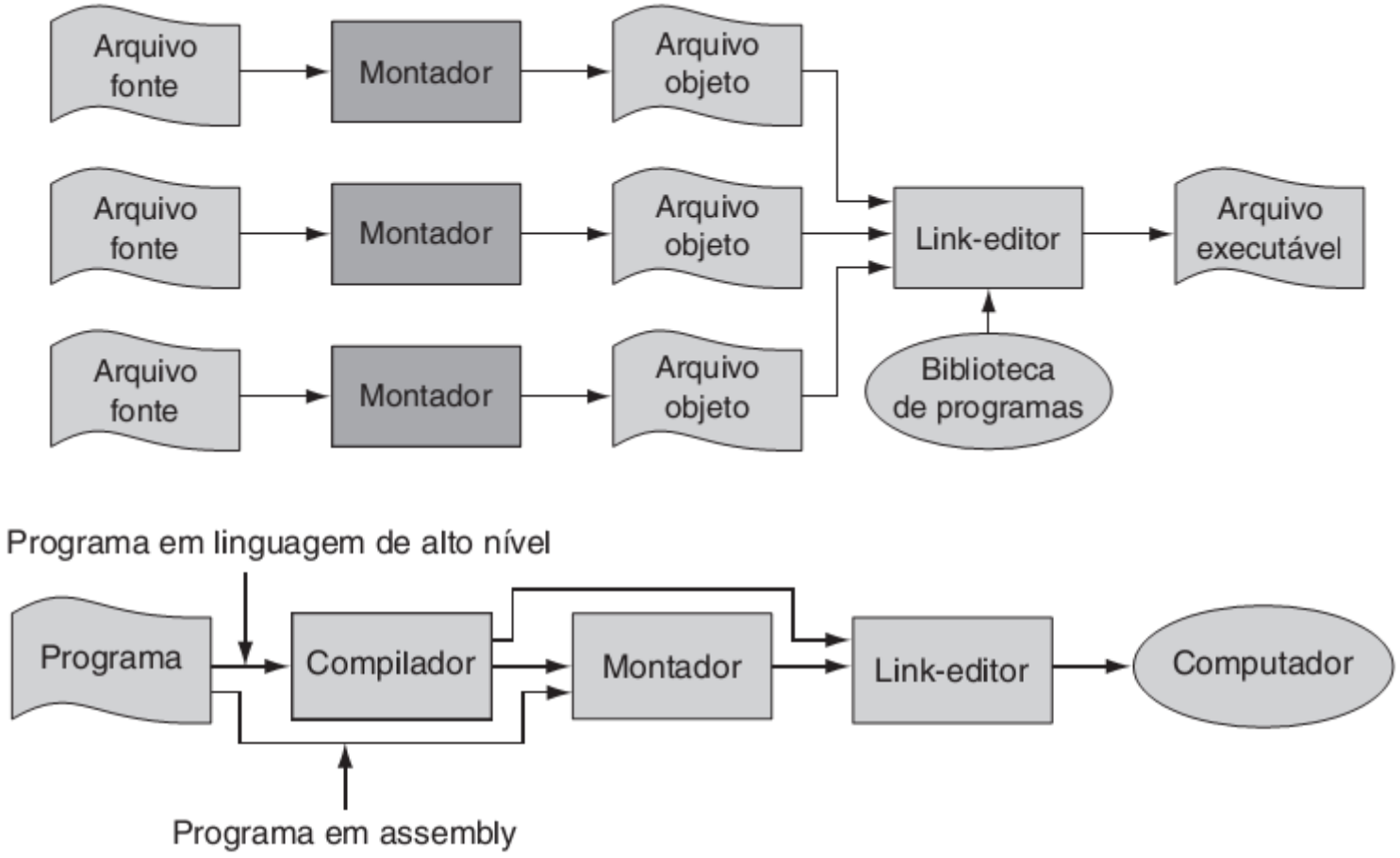
$s3	$s2
A	h

```
lw $t0, 32($s3)    # reg. $t0 recebe A[8]
add $t0, $s2, $t0  # reg. $t0 recebe h+A[8]
sw $t0, 48($s3)    # A[12] recebe h+A[8]
```

Instruções de transferências de dados

Categoria	Nome	Exemplo	Operação	Comentários
Transferência de dados	mfhi	mfhi $8	$8 = HI	
	mflo	mflo $8	$8 = LO	
	lw	lw $8, 4($9)	$8 = MEM[4 + $9]	
	sw	sw $8, 4($9)	MEM[4 + $9] = $8	
	lui	lui $8, 100	$8 = 100 x 2^{16}	Carrega constante na porção alta do registrador de destino. Zera a parte baixa.

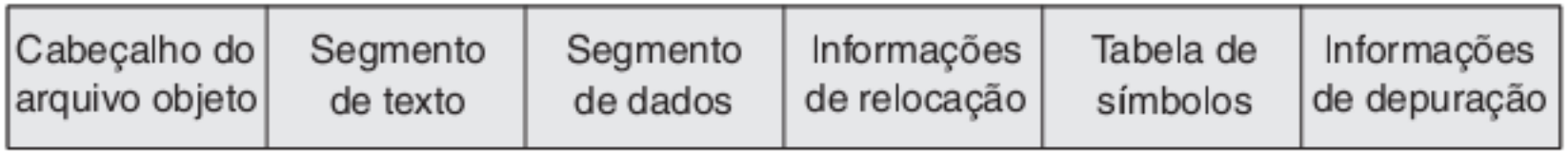
Processo para gerar executável



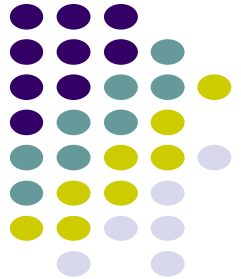
Fonte: PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.

Processo para gerar executável

- Programa objeto dividido em segmentos/seções
  - Um montador Unix gera prog. objeto em 6 segmentos



Diretivas do Assembler



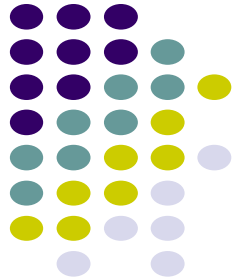
- Facilidades adicionais
  .asciiz "The sum from 0 .. 100 is %d\n"

  - É equivalente a

  .byte 84, 104, 101, 32, 115, 117, 109, 32
  .byte 102, 114, 111, 109, 32, 48, 32, 46
  .byte 46, 32, 49, 48, 48, 32, 105, 115
  .byte 32, 37, 100, 10, 0

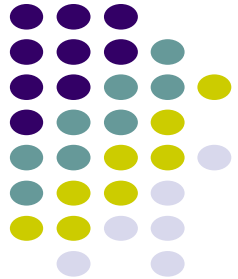
  - Testem no simulador!

Diretivas do Assembler



Sintaxe	Efeito
.ascii str	Armazena string str em memória, mas não coloca o caracter nulo no fim da string.
.asciiz str	Armazena string str em memória, colocando o caracter nulo no fim da string.
.byte b1,..., bn	Armazena as n quantidades de 8 bits em bytes sucessivos na memória
.data <addr>	Itens subsequentes são armazenados no segmento de dados. Se o argumento opcional addr está presente, os itens são armazenados começando no endereço addr.
.global sym	Declara que o símbolo sym é global e pode ser referenciado noutros ficheiros.
.half hl, ..., hn	Armazena n quantidades de 16 bits em sucessivas posições de memória.
.word w1, ...wn	Armazena n quantidades de 32 bits em words sucessivas de memória.
.space n	Reserva n bytes de espaço no segmento de dados.
.text <addr>	Itens subsequentes são colocados no segmento de código. Estando presente o argumento opcional addr, os itens são armazenados a partir do endereço addr.

Chamadas de Sistema



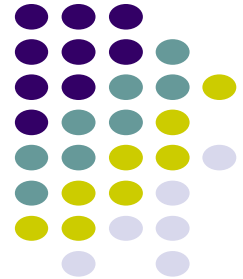
- Estes serviços podem ser executados através da instrução syscall.
- O código da chamada é colocado em **$v0**

Chamadas de Sistema

Códigos para **$v0**

Serviço	Código de chamada do sistema	Argumentos	Resultado
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (em $v0)
read_float	6		float (em $f0)
read_double	7		double (em $f0)
read_string	8	$a0 = buffer, $a1 = tamanho	
sbrk	9	$a0 = valor	endereço (em $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (em $a0)
open	13	$a0 = nome de arquivo (string), $a1 = flags, $a2 = modo	descritor de arquivo (em $a0)
read	14	$a0 = descritor de arquivo, $a1 = buffer, $a2 = tamanho	número de caracteres lidos (em $a0)
write	15	$a0 = descritor de arquivo, $a1 = buffer, $a2 = tamanho	número de caracteres escritos (em $a0)
close	16	$a0 = descritor de arquivo	
exit2	17	$a0 = resultado	

Exemplo



- Teste no simulador MARS um “Hello World”

```
.data
    msg: .asciiz "Ola mundo!"

.text
    la $a0, msg

    addi $v0, $zero, 4
    syscall
```

Exemplo

Seção de dados: declaração de variáveis

Tipo da variável (word) e valor inicial 5

Seção de texto: código fonte

```
# Calcula f = (g+h)-(i+j)
        .data
g: .word   5 # valor 5 armazenado em g
h: .word   2 # valor 2 armanazenado em h
i: .word   1 # valor 1 armanazenado em i
j: .word   3 # valor 3 armanazenado em j
f: .word   0 # valor 0 armanazenado em f

        .text
        # carrengando os valores da memoria
        la   $t0, g          # carrega o endereco de g em $t0
        lw   $s1, 0($t0)     # carrega o valor de g em $s1
        la   $t0, h          # carrega o endereco de h em $t0
        lw   $s2, 0($t0)     # carrega o valor de h em $s2
        la   $t0, i          # carrega o endereco
        lw   $s3, 0($t0)     # carrega o valor de
        la   $t0, j          # carrega o endereco
        lw   $s4, 0($t0)     # carrega o valor de
```

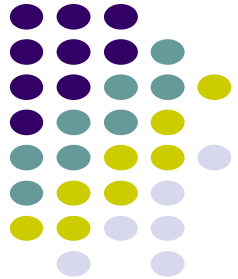
```
#Realizando o calculo
add $t0, $s1, $s2 #soma g e h
add $t1, $s3, $s4 #soma i e j
sub $s0, $t0, $t1 #(g+h)-(i+j)

#Armazena o resultado na memoria
la   $t0, f          # carrega o endereco de f em $t0
sw $s0, 0($t0)       # armazena o valor

# Terminando o programa
li    $v0, 10              # system call for exit
syscall                    # we are out of here.
```

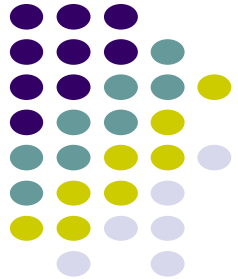
Chamada de sistema: desvia para o SO executar

Exercícios – programas em assembly MIPS



1. Declare duas variáveis inteiras **X** e **Y** inicializadas. Carregue os valores de **X** e **Y** para dois registradores**.** Em seguida incremente em 1 o valor de **X** e armazene resultado em **Y**. verifique no MARS, na execução passo-a-passo as posições de memória das variáveis e os valores sendo atribuídos aos registradores e posições de memória. Para leitura e escrita das variáveis utilize LW e SW.
2. Execute o mesmo programa agora com as variáveis não inicializadas. Então, antes de realizar as operações aritméticas, atribua o valor ao registrador de **X** por meio do teclado.

Exercícios – programas em assembly MIPS



- Em todos os exercícios imprimir o resultado final

1. Programa que realize **a = (b-3)+c** onde **b** e **c** são variáveis inicializadas
2. Programa que leia (por meio de *syscall*) o valor de 3 variáveis (fornecidas pelo usuário), realize a operação **a = (b+c) –(c+d)**
3. Programa que calcula **a = b << 2** e **c = b + b + b + b**, em seguida compare os valores de **a** e **c.**
4. Programa para realizar a expressão **a = b + c – (d – e)** sem precisar utilizar mais de 5 registradores.

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. **Organização e projeto de computadores** – a interface hardware software. 3. ed. Editora Campus, 2005.
- Notas de aula do Prof. André Luis Meneses Silva
- WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005