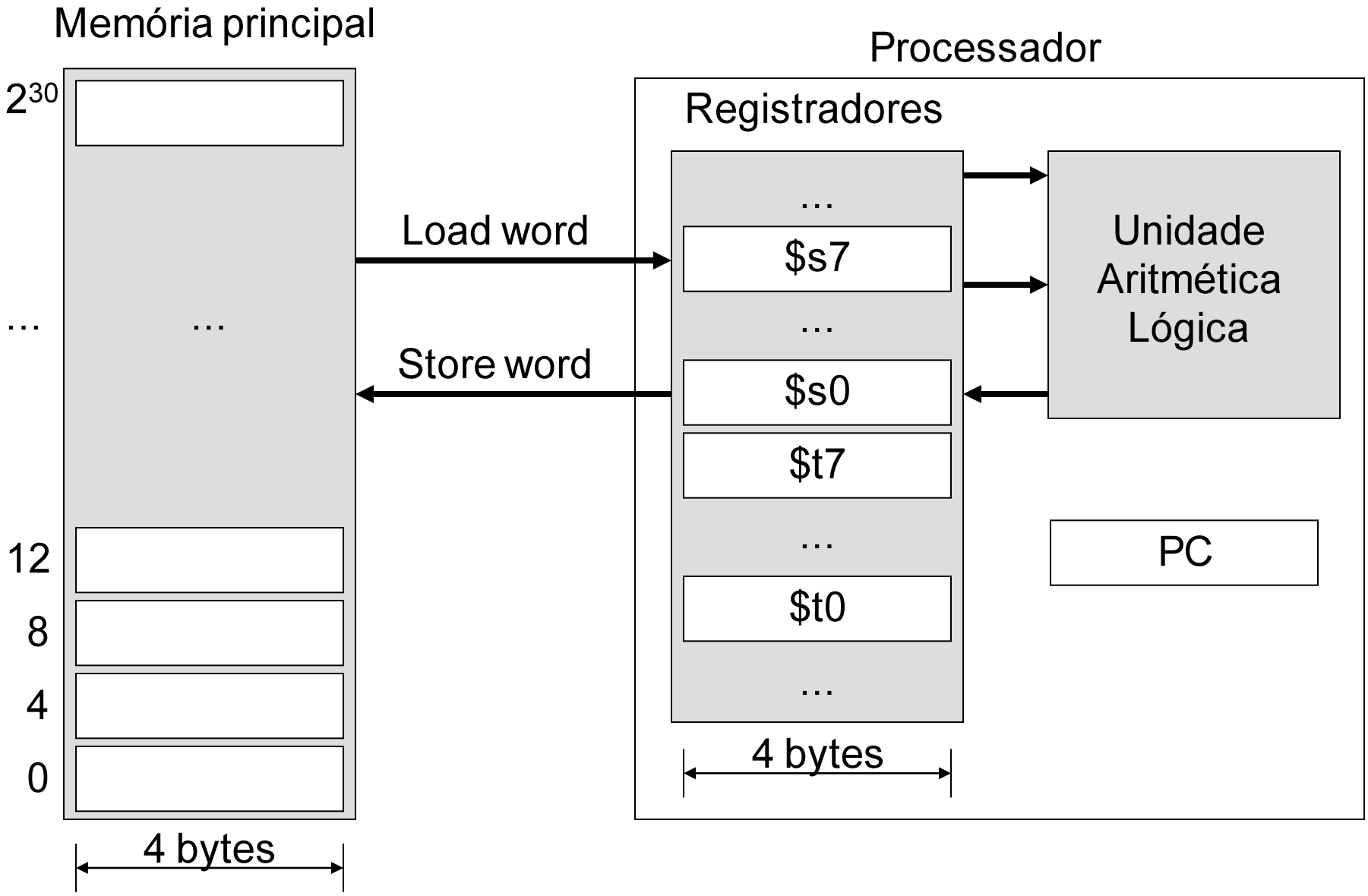
Arquitetura do Processador MIPS
- Programação -

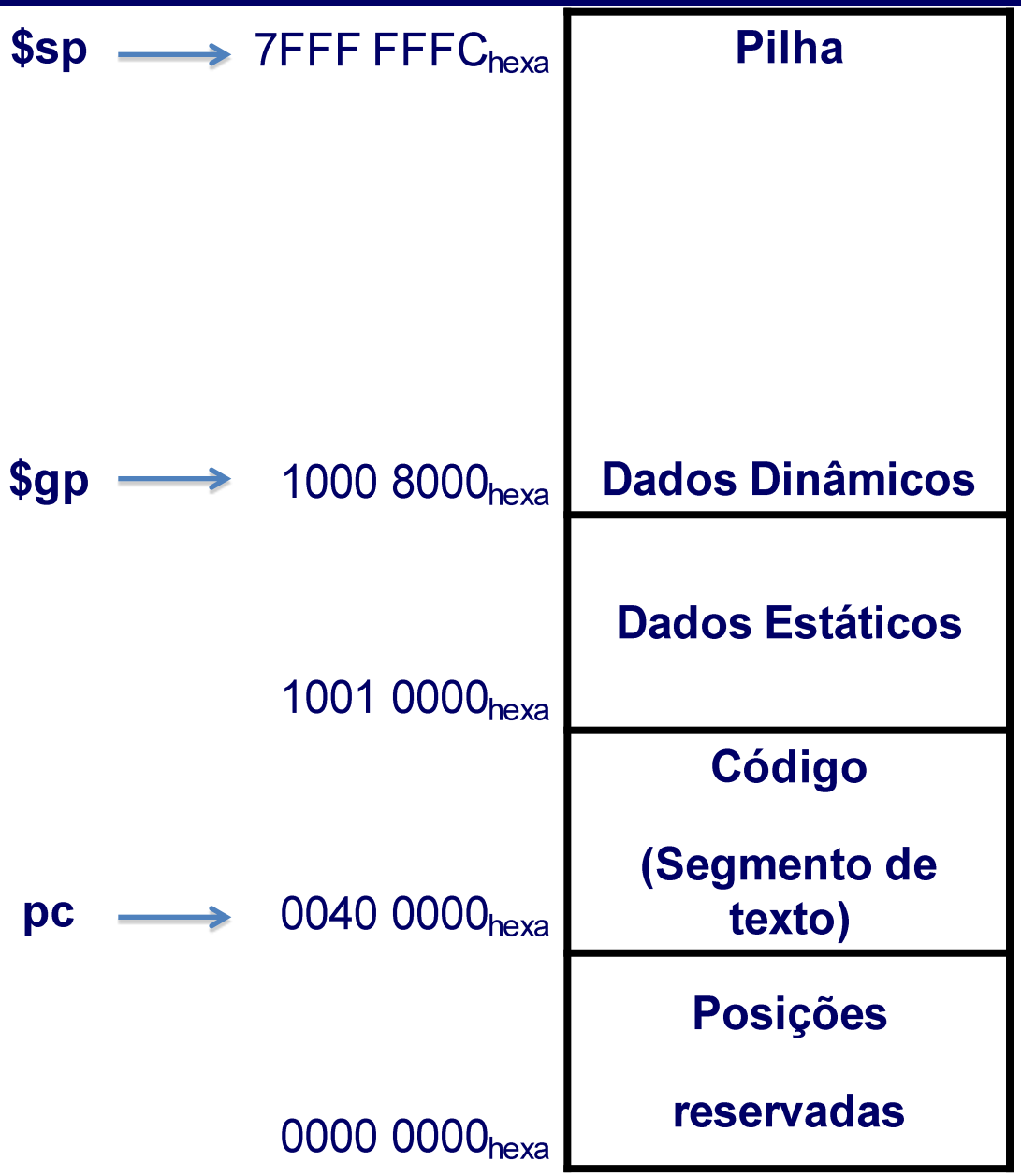
Resumo da arquitetura do MIPS

- Palavra de dados de 32 bits
- Palavra de instrução de 32 bits
- 3 formatos de instrução
- 5 modos de endereçamento
- 32 registradores de uso geral
- Sub-conjunto básico de instruções
  - Aritméticas — add, addi, sub
  - Lógicas — and, or, xor, andi, ori, xori
  - Memória — lw, sw
  - Comparação — slt, slti
  - Desvio condicional — beq, bne
  - Desvio incondicional — j, jal, jr

Visão simplificada do sistema “memória + CPU”



Espaço de endereçamento em memória

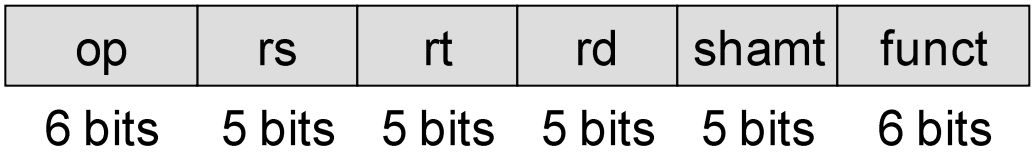


Registradores do MIPS

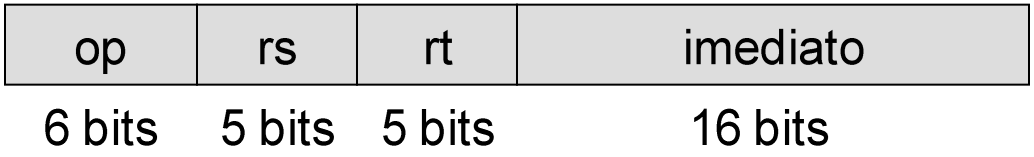
End	Nome	End	Nome	End	Nome	End	Nome
0	$zero	8	$t0	16	$s0	24	$t8
1	$at	9	$t1	17	$s1	25	$t9
2	$v0	10	$t2	18	$s2	26	$k0
3	$v1	11	$t3	19	$s3	27	$k1
4	$a0	12	$t4	20	$s4	28	$gp
5	$a1	13	$t5	21	$s5	29	$sp
6	$a2	14	$t6	22	$s6	30	$fp
7	$a3	15	$t7	23	$s7	31	$ra

Formatos de instrução

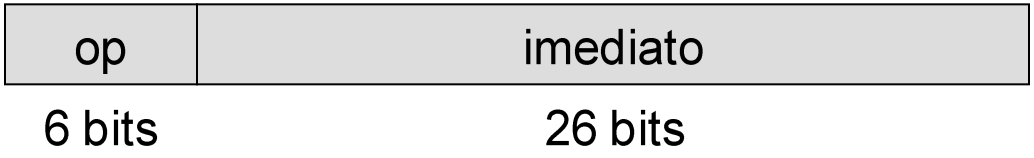
Formato R (registrador)



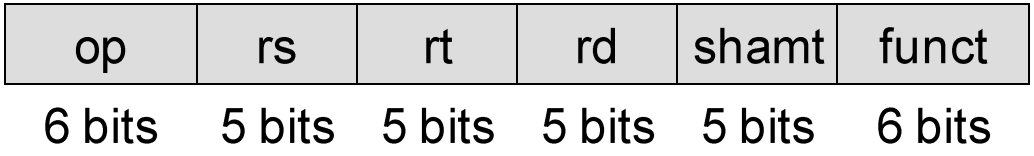
Formato I (imediato)



Formato J (*jump*)

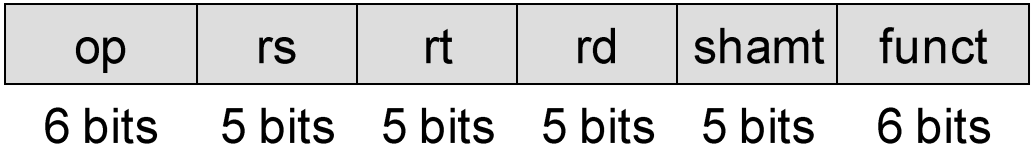


Conjunto de instruções básico: Formato R



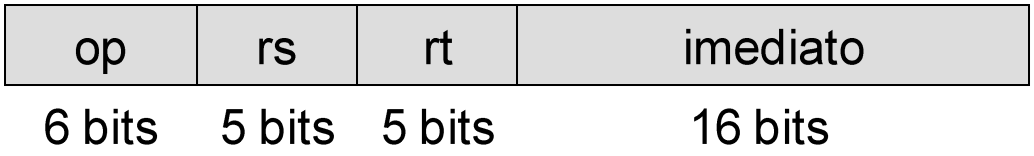
- `add  rd, rs, rt   # rd <- rs + rt`
- `sub  rd, rs, rt   # rd <- rs - rt`
- `and  rd, rs, rt   # rd <- rs & rt`
- `or   rd, rs, rt   # rd <- rs | rt`
- `xor  rd, rs, rt   # rd <- rs ^ rt`
- `nor  rd, rs, rt   # rd <- ~(rs | rt)`

Conjunto de instruções básico: Formato R



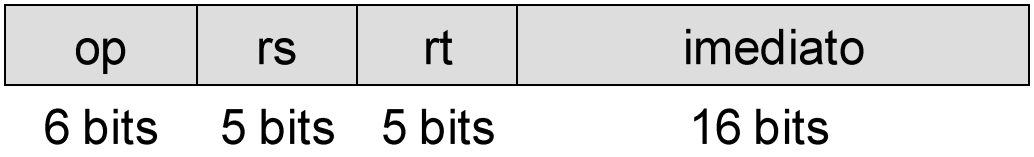
- `sll rd, rs, shamt # rd <- rt << shamt`
- `srl rd, rs, shamt # rd <- rt >> shamt`
- `slt rd, rs, rt # rd <- 1, se rs < rt`
  `# 0, se não`
- `jr rs # PC <- rs`

Conjunto de instruções básico: Formato I



- `addi  rt, rs, imed # rt <- rs + imed`
- `andi  rt, rs, imed # rt <- rs & imed`
- `ori   rt, rs, imed # rt <- rs | imed`
- `xori  rt, rs, imed # rt <- rs ^ imed`
- `slti  rt, rs, imed # rt <- 1,se rs<imed`
  `#          0,se não`

Conjunto de instruções básico: Formato I



- `lw  rt, imed (rs)  # rt <- Mem[imed+rs]`
- `sw  rt, imed (rs)  # Mem[imed+rs] <- rt`
- `beq rs, rt, imed  # Se (rs = rt) então`
  `# PC <- PC+(imed<<2)`
- `bne rs, rt, imed  # Se (rs != rt) então`
  `# PC <- PC+(imed<<2)`

Conjunto de instruções básico: Formato J

6 bits 26 bits

- `j    imed  # PC  <- PC[31..28]:(imed<<2)`
- `jal  imed  # PC  <- PC[31..28]:(imed<<2)`
  `# $ra <- PC ou $ra <- PC+4`

Pseudo-instruções

- **Instruções que o montador converte em instruções do processador**
- **Pseudo-instruções de desvio condicional**
  - Desvia se igual a zero `beqz rsrc,label`
  - Desvia se maior que `bgt rsrc1,rsrc2,label`
  - Desvia se maior ou igual que `bge rsrc1,rsrc2,label`
  - Desvia se menor que `blt rsrc1,rsrc2,label`
  - Desvia se menor ou igual que `ble rsrc1,rsrc2,label`
  - Desvia se diferente de zero `bnez rsrc,label`
- **Outras pseudo-instruções**
  - Inversão `not rdest, rsrc`
  - Carga de vetor `la rdest, vetor`
  - Carga de constante `li rdest, cte`
  - Carga de registrador `move rdest, rsrc`

Estrutura do código

- **Código não estruturado, organizado em colunas**

Rótulo:
    Mnemônico Lista_de_Operandos # Comentários

- **Segmento de dados**
  - Onde são declaradas as variáveis do programa
  - Demarcado pela diretiva `.data`
  - Diretivas de dados
    - `.word` : declaração de inteiros e vetores de inteiros
    - `.asciiz` : declaração de strings

- **Segmento de código**
  - Instruções do programa
  - Demarcado pela diretiva `.text`

Estrutura do código

```
        .data
ex_int : .word    5
ex_vec : .word    10, 20, 30, 40, 50
ex_str : .asciiz  "\nHello world!"
ex_buf : .asciiz  ""
```

```
        .text
main :
        addi $t0, $zero, 2    # $t0 = 2
        addi $t1, $zero, 2    # $t1 = 2
        add  $t2, $t0, $t1    # $t2 = $t0+$t1
```

A instrução syscall

- **Serviços de interface**
- **Uso em serviços (com inteiros e strings)**
  - Carregar código do serviço em $v0
  - Carregar argumento (se houver) em $a*i*
  - Executar a instrução syscall
  - Ler o valor de retorno (se houver) de $v0
- **Principais serviços (com inteiros e strings)**

Serviço	Código ($v0)	Argumentos ($a*i*)	Retorno ($v0)
Print_int	1	$a0 : Inteiro	
Print_string	4	$a0 : Ponteiro para string	
Read_int	5		Inteiro
Read_string	8	$a0 : Ponteiro para buffer $a1 : Tamanho bo buffer	
Exit	10		

A instrução syscall

- **Leitura de um número inteiro**

```
                        # READ_INT
addi $v0, $0, 5         # Carrega serviço 5
syscall                 # Chama o serviço
add  $s0, $0, $v0       # Carrega retorno em $s0
```

Nota: $0 = $zero = 0

ou

```
                        # READ_INT
li   $v0, 5             # Carrega serviço 5
syscall                 # Chama o serviço
move $s0, $v0           # Carrega retorno em $s0
```

A instrução syscall

- **Impressão de um número inteiro**

```
                        # PRINT_INT
addi $v0, $0, 1         # Carrega serviço 1
add  $a0, $0, $s0       # Carrega int em $a0
syscall                 # Chama o serviço
```

ou

```
                        # READ_INT
li   $v0, 1             # Carrega serviço 1
move $a0, $s0           # Carrega int em $a0
syscall                 # Chama o serviço
```

A instrução syscall

- **Leitura de uma string**
  - Considere a existência de um buffer com capacidade de armazenar 16 caracteres

```
                        # READ_STRING
li    $v0, 8            # Carrega serviço 8
la    $a0, buffer       # Carrega ptr p/ buffer
li    $a1, 16           # Carrega tam. máximo
syscall                 # Chama o serviço
                        # A string lida está
                        # armazenada em buffer
```

A instrução syscall

- **Impressão de uma string**
  - Considere uma string declarada como message

```
                        # PRINT_STRING
li   $v0, 4             # Carrega serviço 4
la   $a0, message       # Carrega ptr p/ string
syscall                 # Chama o serviço
```