Sistemas Processadores e Periféricos
Aula 3 - Revisão

Prof. Frank Sill Torres
DELT – Escola de Engenharia
UFMG

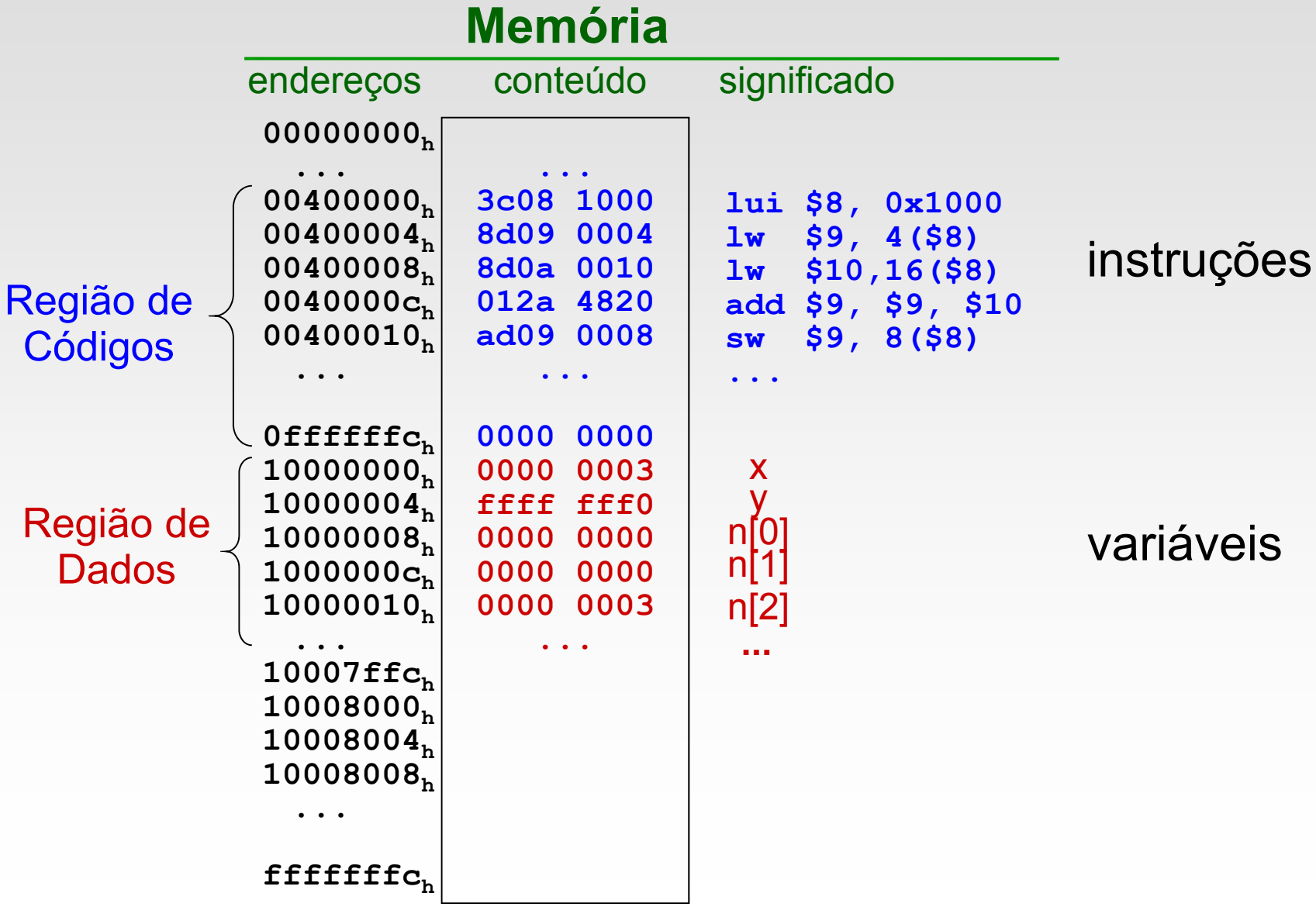
Adaptado a partir dos Slides de Organização de Computadores 2006/02 do professor

Leandro Galvão DCC/UFAM - galvao@dcc.ufam.edu.br pelo Prof. Ricardo de Oliveira Duarte

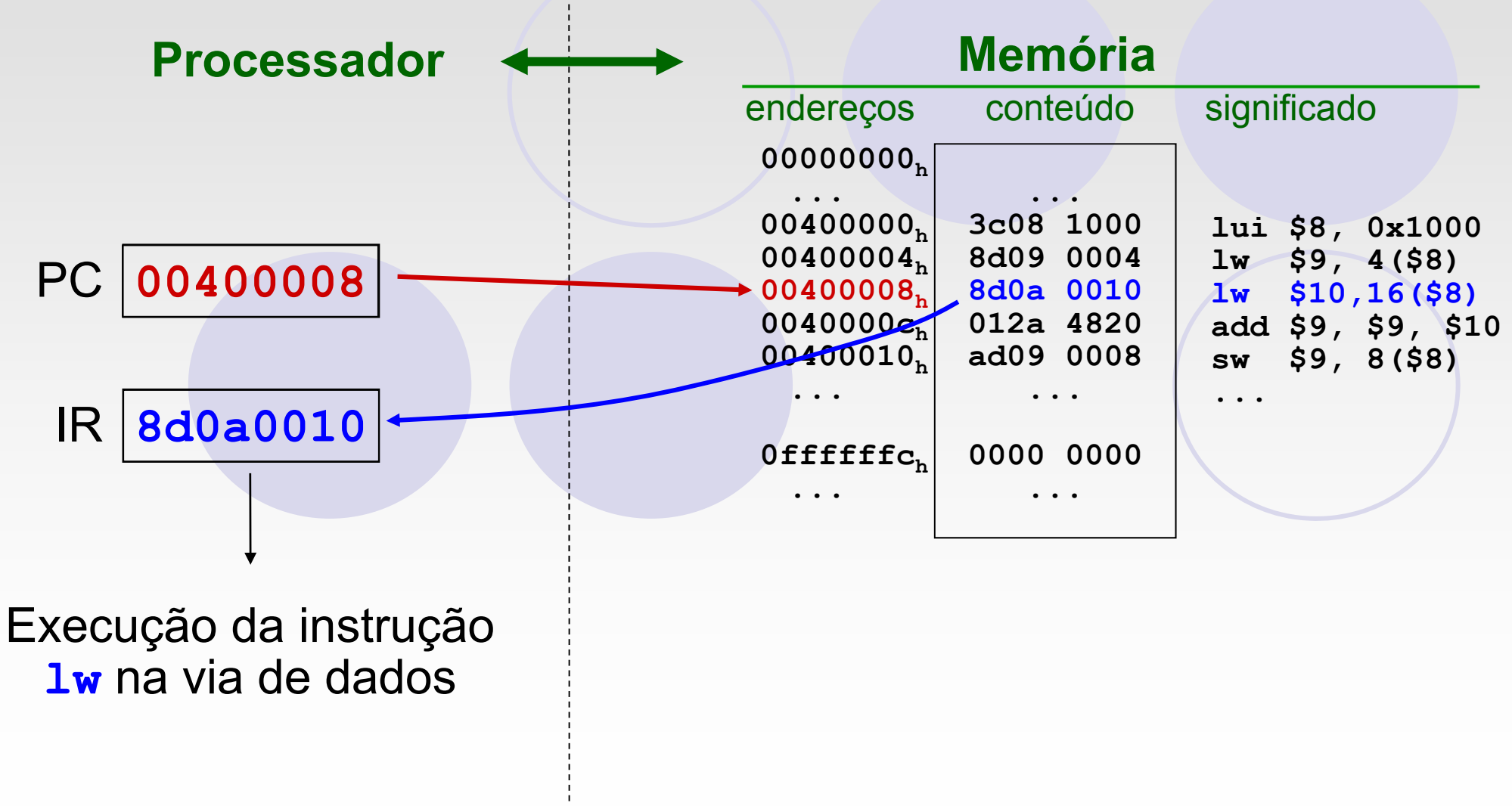
Instruções MIPS
:: Instruções Lógicas

Operação lógica	Instrução MIPS	Significado
Shift à esquerda	`sll`	*Shift left logical*
Shift à direita	`srl`	*Shift right logical*
AND bit a bit	`and, andi`	*E*
OR bit a bit	`or, ori`	*OR*
NOT bit a bit	`nor`	*NOR*

Instruções MIPS
:: Processando instruções sequenciais



Instruções MIPS
:: Processando instruções seqüenciais



Instruções MIPS
:: Instruções de controle

- **Branch on not equal (`bne`)**
  - Desvia o programa para <label1> se $t0 != $t1

```
bne  $t0, $t1, label1  #if ($t0 != $t1) goto label1
```

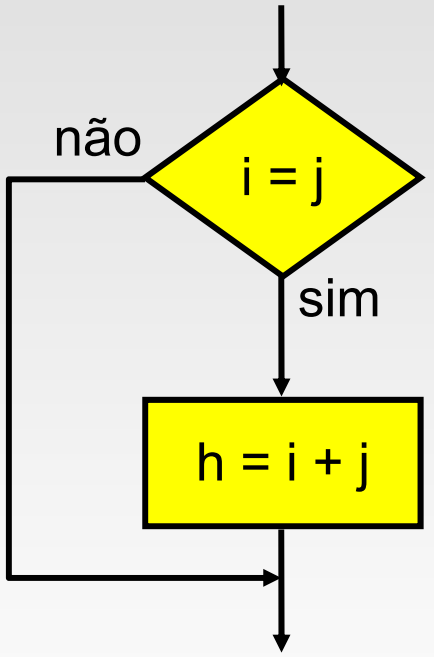
- **Branch on equal (`beq`)**
  - Desvia o programa para <label2> se $t0 == $t1

```
beq  $t0, $t1, label2  #if ($t0 == $t1) goto label2
```

Instruções MIPS
:: Instruções de controle :: Ex 01

- Exemplo

```
      bne $8, $9, sai
      add $10, $8, $9
sai:    nop
```



Instruções MIPS
:: Instruções de controle

- **Set on less than (`slt`)**
  - Compara dois registradores

```
slt   $s1, $s2, $s3     #if ($s2 < $s3) $s1 = 1
                        #else $s1 = 0
```

- **Set on less than immediate (`slti`)**
  - Compara um registrador e uma constante

```
slti  $s1, $s2, 100     #if ($s2 < 100) $s1 = 1
                        #else $s1 = 0
```

Instruções MIPS
:: Instruções de controle

- **Jump (`j`)**
  - Desvio incondicional para um endereço de memória apontado por um label

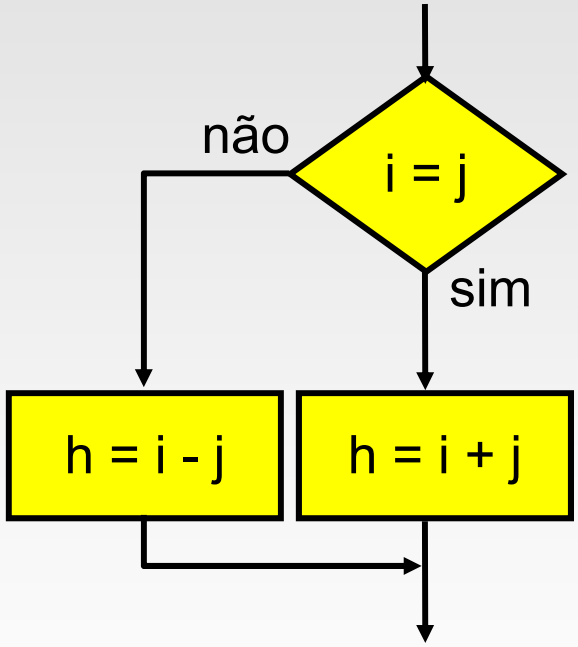
```
j   label
```

  - Instruções do tipo branch indicam desvio da sequência do programa mediante análise de uma condição
  - Instruções do tipo jump indicam desvio incondicional da sequência do programa

Instruções MIPS
:: Instruções de controle :: Ex 02

- Exemplo

```
        bne  $8, $9, else
        add  $10, $8, $9
        j    sai
else:   sub  $10, $8, $9
sai:    nop
```



Instruções MIPS
:: Instruções de controle :: Formato

- Instruções BNE e BEQ:
  - Campo “immediate” possui quantidade de palavras (words) que devem ser saltadas para chegar ‘a instrução marcada pelo label (rótulo)
  - O Número pode ser positivo (desvio para frente) e negativo (desvio para trás)
- Instrução J (jump):
  - Definição do endereço da memória correspondente à instrução marcada pelo label
  - Novo endereço: 4 MSB do PC atual + 26 Bits da instrução deslocado a esquerda de 2 Bits (como o endereço da memória é um múltiplo de 4 bytes)

Sistemas Processadores e Periféricos

Aula 4 - Conjunto de Instruções MIPS III

Prof. Frank Sill Torres
DELT – Escola de Engenharia
UFMG

Adaptado a partir dos Slides de Organização de Computadores 2006/02 do professor

Leandro Galvão DCC/UFAM - galvao@dcc.ufam.edu.br pelo Prof. Ricardo de Oliveira Duarte

Instruções MIPS

- Transferência de Dados
- Lógicas
- Controle
- Suporte a procedimentos

Instruções MIPS
:: Suporte a Procedimentos

- Procedimentos: Conjunto de instruções com função definida
- Realizam uma série de operações como base em valores de parâmetros
- Podem retornar valores computados

Instruções MIPS
:: Suporte a Procedimentos

- Motivos para o uso de procedimentos:
  - Tornar o programa mais fácil de ser entendido
  - Permitir a reutilização do código do procedimento
  - Permitir que o programador se concentre em uma parte do código (os parâmetros funcionam como barreira)

Instruções MIPS
:: Suporte a Procedimentos

- Passos para a execução:
  1. O programa coloca os parâmetros em um lugar onde o procedimento chamado possa acessá-los
  2. O programa transfere o controle para o procedimento
  3. O procedimento acessa os valores necessários à realização de sua tarefa
  4. O procedimento executa sua tarefa, gerando valores
  5. O procedimento (chamado) coloca os valores gerados em um lugar onde o programa (chamador) pode acessá-los
  6. O procedimento transfere o controle de volta para o ponto do programa que o chamou

Instruções MIPS
:: Suporte a Procedimentos

- Característica dos procedimentos típicos
  - Têm poucos parâmetros
  - Retornam poucos parâmetros, muitas vezes um único

- Parâmetros podem ser passados em registradores

Instruções MIPS
:: Suporte a Procedimentos

- MIPS aloca alguns registradores para implementar o uso de procedimentos:
  - **`$a0-$a3`**

    Quatro registradores usados como argumentos para passagem de parâmetros
  - **`$v0-$v1`**

    Dois registradores usados para retornar valores
  - **`$ra`**

    Registrador que guarda o endereço de retorno (*return address*) para o ponto do programa que chamou o procedimento

Instruções MIPS
:: Suporte a Procedimentos

Instruções MIPS
:: Suporte a Procedimentos

- **Jump and link (`jal`)**
  - Salta para o endereço especificado, salvando o endereço da próxima instrução em **`$ra`**

```
jal   label  #desvia para o endereço indicado
             #por label.  $ra ← PC + 4
```

- **Jump register (`jr`)**
  - Desvio incondicional para endereço guardado em **`$ra`**

```
jr  $ra      #desvia para o endereço da
             #memória guardado em $ra
```

Instruções MIPS
:: Suporte a Procedimentos

- Passos no MIPS para execução de procedimentos
  1. Chamador coloca os valores dos parâmetros em `$a0-$a3`
  2. Chamador chama `jal X` para saltar ao procedimento X (chamado)
  3. Chamado realiza suas tarefas
  4. Chamado coloca os resultados em `$v0-$v1`
  5. Chamado retorna o controle ao chamador usando `jr $ra`

Usando mais registradores

- Se forem necessários mais que quatro argumentos e/ou dois resultados?
  - **`$t0-$t9`**:

    Dez registradores temporários que NÃO SÃO preservados pelo chamador

  - **`$s0-$s7`**:

    Oito registradores que DEVEM ser preservados se utilizados no procedimento chamado

Usando mais registradores

- Registradores "*caller-saved*": **`$t0-$t9`**

  Chamador é responsável por salvá-los em memória caso seja necessário usá-los novamente depois que um procedimento é chamado

- Registradores "*callee-saved*": **`$s0-$s7`**

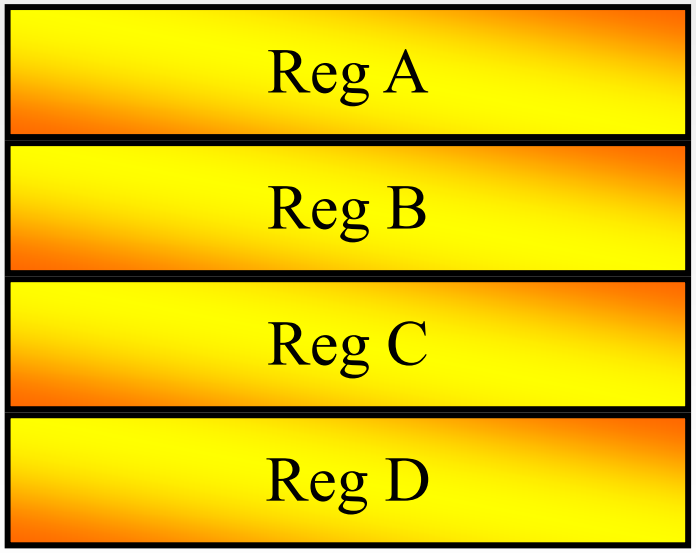
  Chamado é responsável por salvá-los em memória antes de utilizá-los e restaurá-los antes de devolver o controle ao chamador

Preservação de Contexto

- Qualquer registrador usado pelo chamador deve ter seu conteúdo restaurado para o valor que tinha antes da chamada
- Conteúdo dos registradores é salvo na memória. Depois da execução do procedimento, estes registradores devem ter seus valores restaurados
- Se as chamadas forem recursivas, é conveniente o uso de uma pilha

Pilha

- Espaço da memória organizada com uma fila to tipo “último a entrar, primeiro a sair” (FILO):



Apontador de Pilha (Stack Pointer)

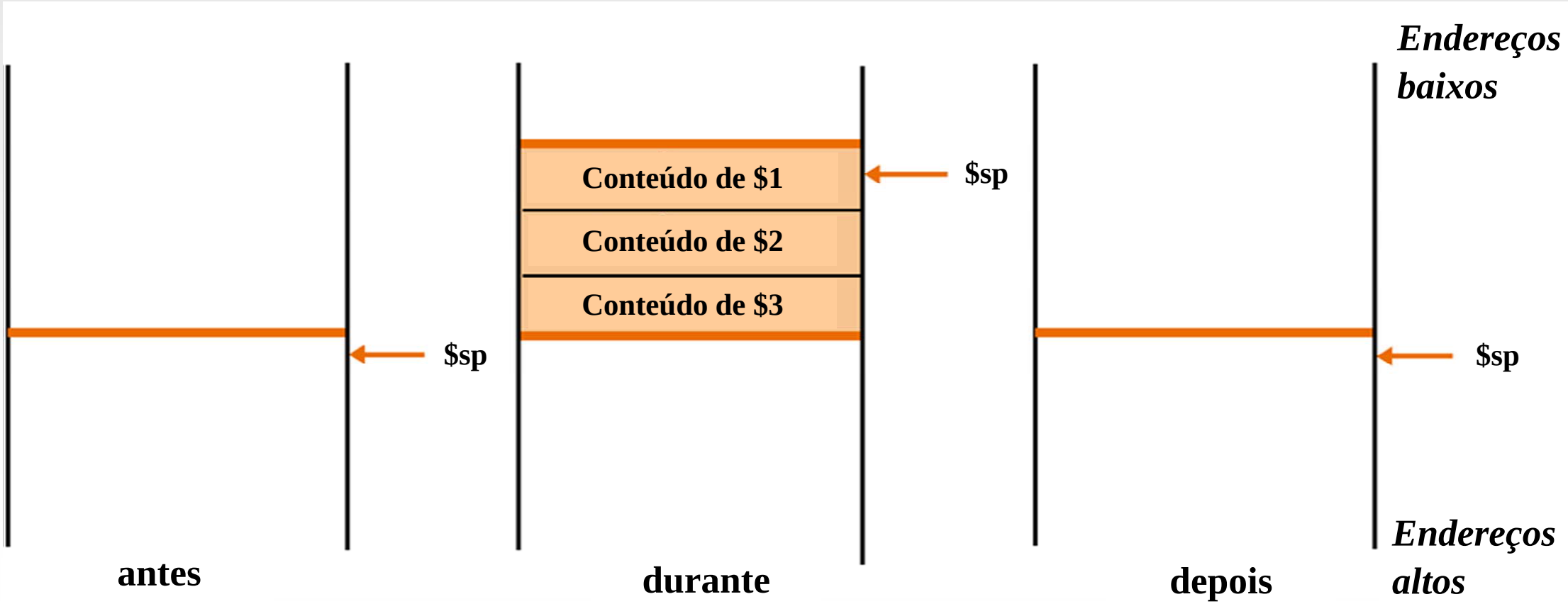
- `$sp`
  - Registrador usado para guardar o endereço do topo da pilha da chamada de procedimentos

- Indica:
  - a posição de memória que contêm os valores dos registradores salvos na memória pela última chamada
  - a posição a partir da qual a próxima chamada de procedimento pode salvar seus registradores

- A pilha cresce do endereço mais alto para o mais baixo

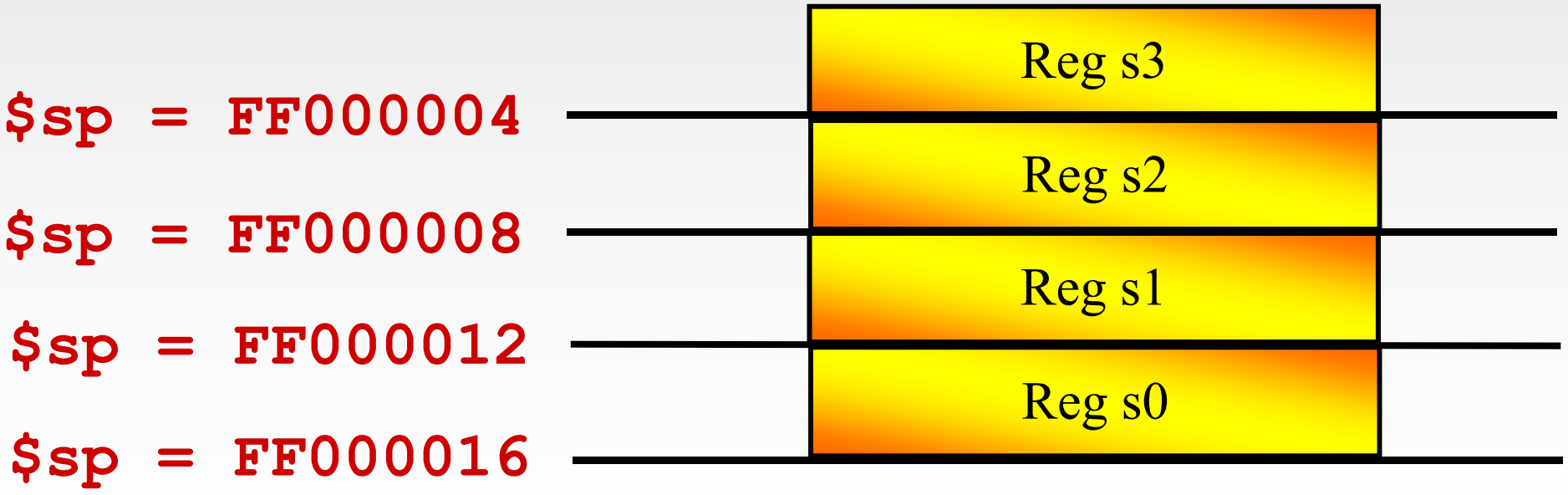
Apontador de Pilha (Stack Pointer)

- Valor do stack pointer (`$sp`) em momentos diferentes da chamada de procedimento:



Pilha

- Exemplo: Valor do stack pointer (**`$sp`**) em momentos diferentes da chamada de procedimento:



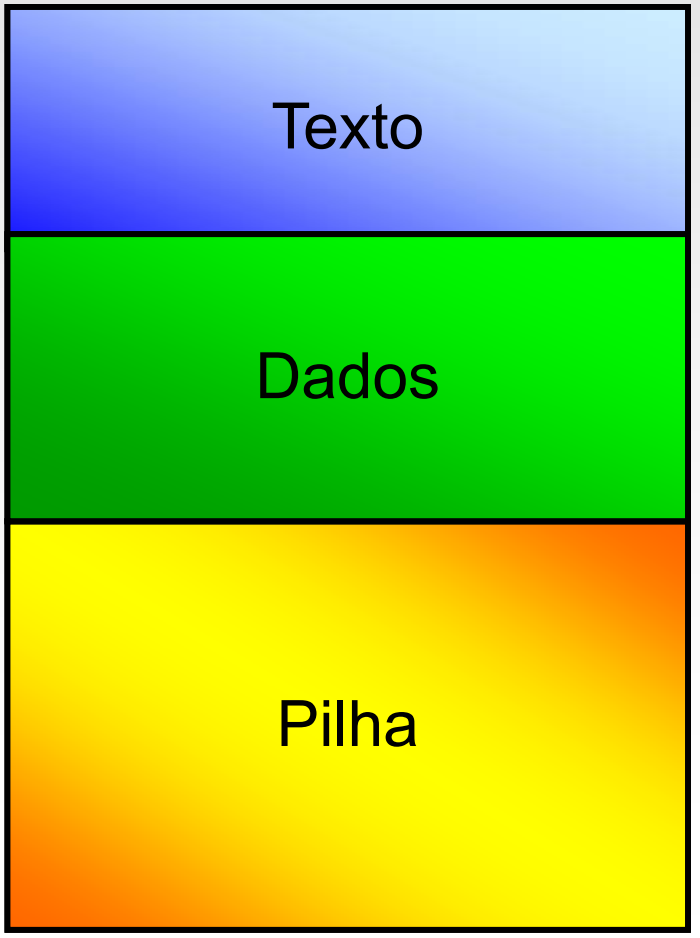
Pilha

Para todo programa, o sistema operacional aloca três segmentos de memória:

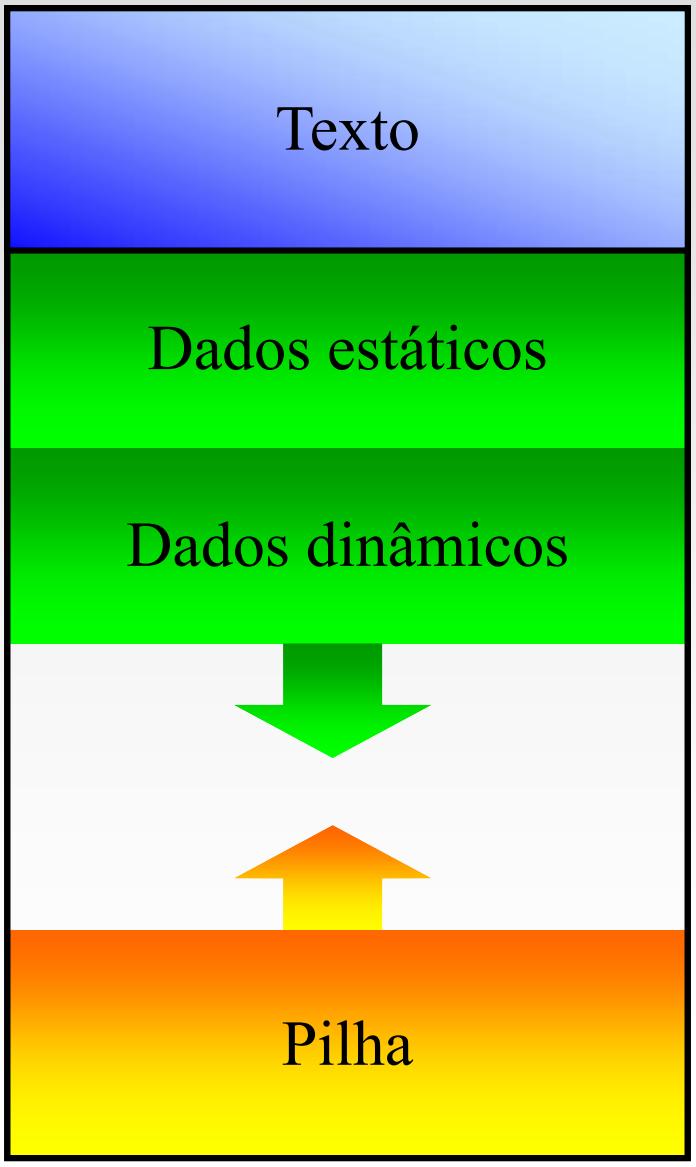
Segmento de texto: armazena o código de máquina

Segmento de dados: alocado para constantes e variáveis globais

Segmento de pilha: local onde são passados parâmetros, alocado espaço para variáveis locais e armazenados endereços de retorno para chamadas de funções aninhadas/recursivas



Pilha



Suporte a Procedimentos :: Exemplo 1

- Compilar o seguinte código C em código assembly do MIPS:

```
int leaf_example(int g, int h, int i, int j)
{
   int f;
   f = (g + h) - (i + j);
   return f;
}
```

Suporte a Procedimentos :: Exemplo 1

```
int leaf_example(int g, int h, int i, int j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

- **$a0** corresponde a **g** , **$a1** corresponde a **h**
- **$a2** corresponde a **i** , **$a3** corresponde a **j**
- **$s0** corresponde a **f**
  - Variável local (deve ser salva, pois será modificada pelo procedimento)
- Valor de retorno deve ser colocado em **$v0**

Suporte a Procedimentos :: Exemplo 1

```
leaf_example:                   # label para chamada
   addi  $sp, $sp, -4           # avança o stack pointer
   sw    $s0, 0($sp)            # empilha o valor de $s0

   add   $t0, $a0, $a1          # $t0 ← g + h
   add   $t1, $a2, $a3          # $t1 ← i + j
   sub   $s0, $t0, $t1          # f ← $t0 - $t1

   add   $v0, $s0, $zero        # coloca resultado em $v0

   lw    $s0, 0($sp)            # restaura $s0
   addi  $sp,$sp,4              # desempilha o topo
   jr    $ra                    # volta para o chamador
```

Suporte a Procedimentos :: Procedimentos Aninhados

- Chamadas Sucessivas:
  - Programa principal chama procedimento **A** com um argumento
  - **A** chama procedimento **B** com um argumento

- Chamadas recursivas:
  - **A** chama **A**

- Possíveis problemas:
  - `$a0` será sobreposto quando **B** é chamado e o valor do parâmetro passado na chamada de **A** será perdido
  - Quando **B** for chamado pela instrução `jal`, o registrador `$ra` será sobreposto

Suporte a Procedimentos :: Procedimentos Aninhados

- O procedimento chamador coloca na pilha todos os registradores de argumento (`$a0-$a3`) e/ou registradores temporários (`$t0-$t9`) necessários após a chamada
- O procedimento chamado salva na pilha o endereço de retorno (`$ra`) e todos os registradores de salvamento usados por ele (`$s0-$s7`)
- O apontador da pilha (`$sp`) é ajustado para acomodar a quantidade de registradores colocados na pilha
- Quando do retorno, os valores dos registradores são restaurados a partir da pilha e `$sp` é atualizado

Suporte a Procedimentos :: Exemplo de Recursão

- Compilar o seguinte código C em código assembly do MIPS:

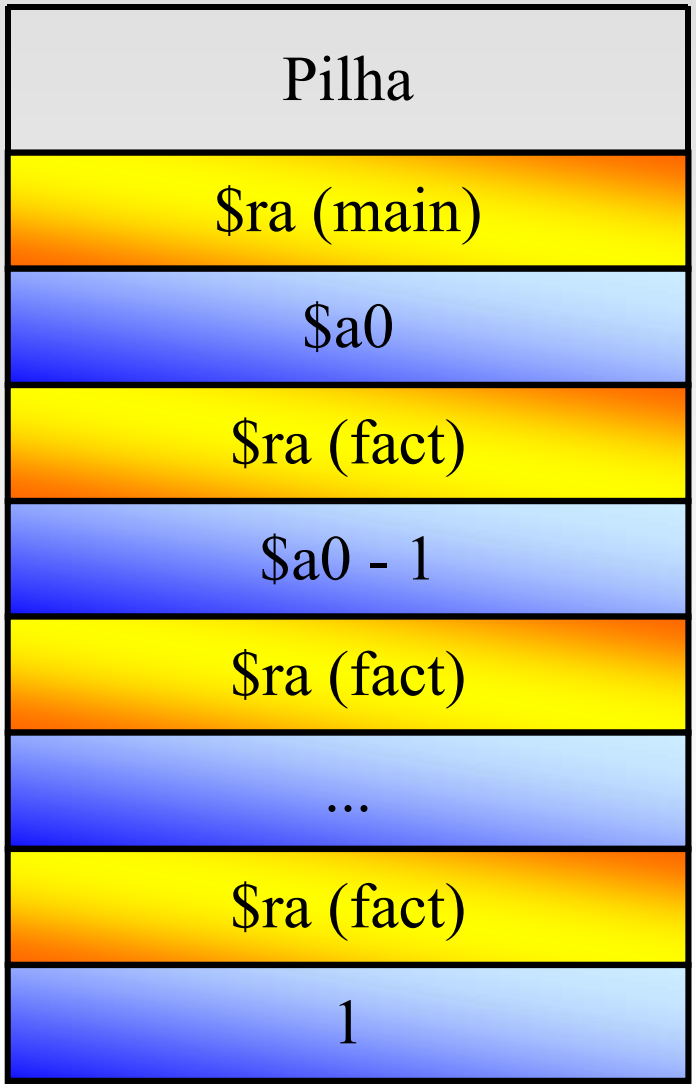
```
z = fact(x);

int fact(int n)
{
if (n < 1)
   return(1);
else
   return(n * fact(n-1));
}
```

Suporte a Procedimentos :: Exemplo de Recursão

- Algoritmo:
  1) Subtrair o valor 1 do registrador `$a0`
  2) Guardar novo valor de `$a0` na pilha
  3) Guardar endereço de retorno (`$ra`)
  4) Repetir 1, 2 e 3 até que `$a0` seja 1
  5) Recuperar da pilha, passo a passo, os diversos valores `$a0`, multiplicando o valor atual do fatorial pelo valor recuperado
  6) Repetir 5 até que o endereço de retorno (`$ra`) seja o da função principal

Suporte a Procedimentos
:: Exemplo de Recursão



$v0*...*($a0)

Suporte a Procedimentos
:: Exemplo de Recursão

```
fact: addi  $sp, $sp, -8       # abre espaço para 2 itens na pilha
      sw    $ra, 4($sp)        # salva o endereço de retorno
      sw    $a0, 0($sp)        # salva o argumento n
      slti  $t0, $a0, 1        # testa se n < 1
      beq   $t0, $zero, L1     # se n >= 1, desvia para L1
      addi  $sp, $sp, 8        # elimina 2 itens da pilha
      addi  $v0, $zero, 1      # retorna o valor 1
      jr    $ra                # retorna para ponto de chamada

L1:   addi  $a0, $a0, -1       # n>=1, o argumento recebe (n-1)
      jal   fact               # chama fact com argumento (n-1)

      lw    $a0, 0($sp)        # restaura argumento n
      lw    $ra, 4($sp)        # restaura o endereço de retorno
      add   $sp, $sp, 8        # ajusta $sp para eliminar 2 itens

      mul   $v0, $a0, $v0      # retorna n*fact(n-1)
      jr    $ra                # retorna para o chamador
```

Suporte a Procedimentos :: Procedimentos Aninhados

Não precisam ser preservados pelo chamador	Se usados, o chamador precisa salvar seus valores e depois restaurá-los
Reg de salvamento: $s0-$s7	Reg temporários: $t0-$t9
Reg stack pointer: $sp	Reg de argumento: $a0-$a3
Reg de endereço de retorno: $ra	Reg de retorno de valores: $v0-$v1
Pilha acima do stack pointer	Pilha abaixo do stack pointer

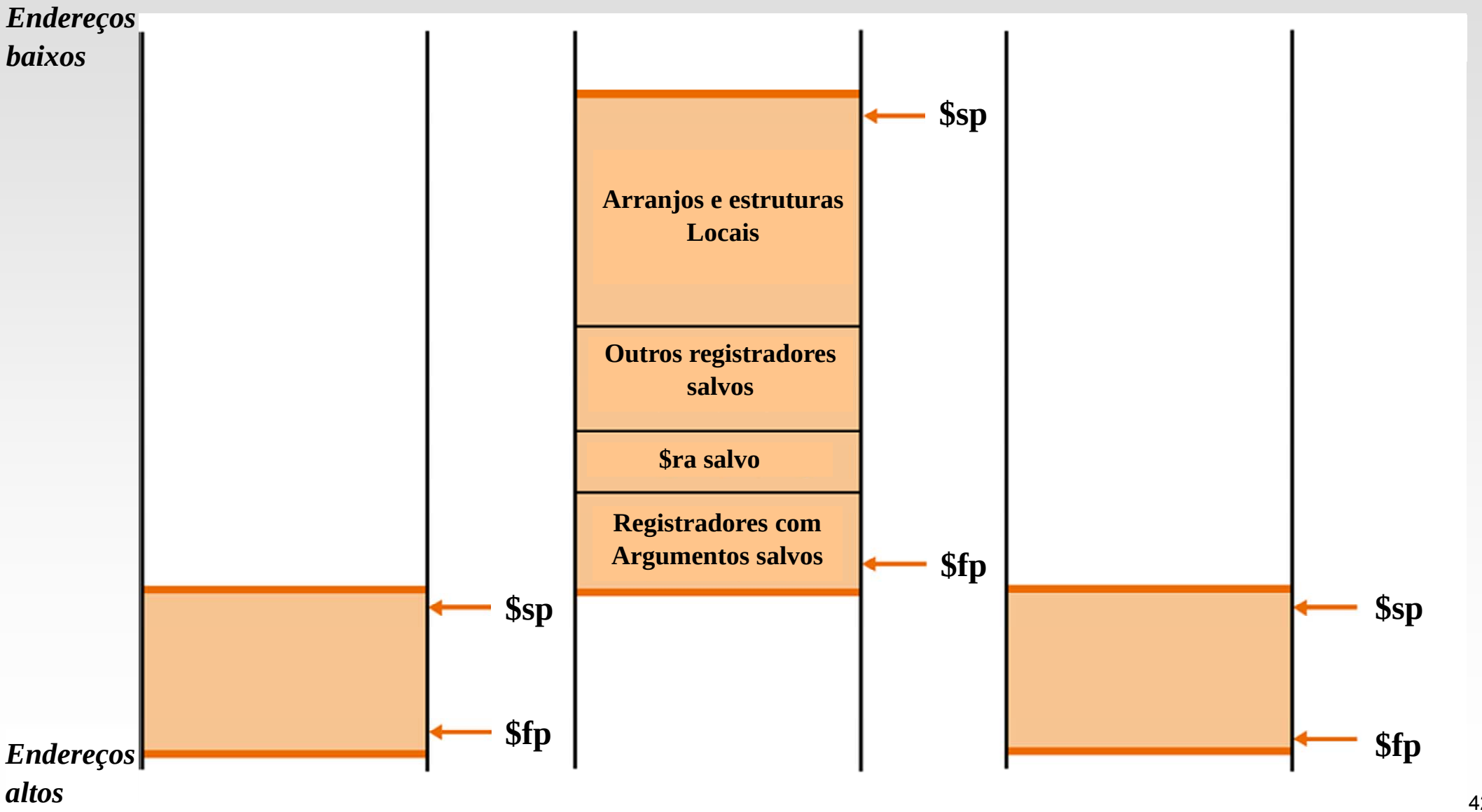
Suporte a Procedimentos :: Quadro de Procedimento

- Quadro de Procedimento (Procedure Frame) ou Registro de Ativação:
  - Segmento da pilha é usado para salvar o conteúdo dos registradores e armazenar variáveis locais
  - Valor de `$sp` indica o “topo” da pilha

Suporte a Procedimentos :: Quadro de Procedimento

- `$fp` - first word procedure frame
  - Pode ser usado para indicar a primeira palavra do quadro de pilha
  - Atua como registrador base estável dentro de um procedimento para referência de memória local
  - Seu uso não é obrigatório, mas é necessário quando `$sp` é alterado durante a execução do procedimento

Quadro de Procedimento
`$sp e $fp`



Quadro de Procedimento :: Ações do Chamador

- Passagem de argumentos
  - Os 4 primeiros são colocados nos registradores `$a0-$a3`
  - O restante é colocado na pilha
    - Outros argumentos armazenados no quadro do procedimento
    - `$sp` aponta para o último argumento

- Registradores *caller-saved*
  - `$a0-$a3` e `$t0-$t9`
  - Salvar se e somente se o chamador precisar do conteúdo intacto depois que a chamada retornar

Quadro de Procedimento :: Ações do Chamado

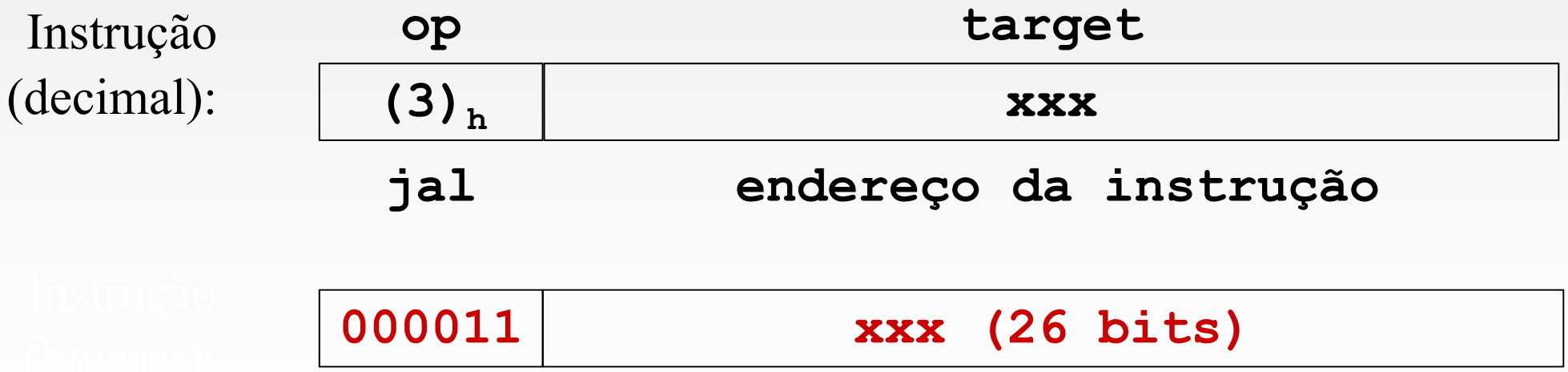
- Alocar memória na pilha:
  - `$sp` ← `$sp - tamanho do quadro`
- Salvar registradores *callee-saved* no quadro antes de alterá-los:
  - `$s0-$s7`, `$fp` e `$ra`
  - O chamador espera encontrá-los intactos depois da chamada
  - Salvar `$fp` a cada procedimento que aloca um novo quadro na pilha
  - Só é necessário salvar `$ra` se o chamado fizer alguma chamada
- Ao final:
  - Retornar o valor em `$v0`
  - Restaurar registradores callee-saved salvos no início da chamada
  - Remover o quadro adicionando o tamanho do quadro a `$sp`
  - Executar `jr $ra`

Instruções MIPS

:: Suporte a Procedimentos :: Formato

- **Jump and link (`jal`)**

```
jal   label    #desvia para o endereço indicado
               #por label. $ra ← PC + 4
```



Instruções MIPS

:: Suporte a Procedimentos :: Formato

- **Jump register (`jr`)**

```
jr   $t3      # PC ← endereço[$t3]
```

Instrução (decimal):

op	rs	rt	rd	shamt	funct
0	11	0	0	0	$(8)_h$
jr	$t3	--	--	--	jr

Instrução (binário):

000000	01011	00000	00000	00000	001000

Instruções MIPS
:: Suporte a Procedimentos :: Resumo

Categoria	Nome	Exemplo	Operação
Suporte a procedimentos	jal	jal rotulo	$31 ← endereço[retorno] PC ← endereço[rotulo]
	jr	jr $31	PC ← $31

Instruções MIPS

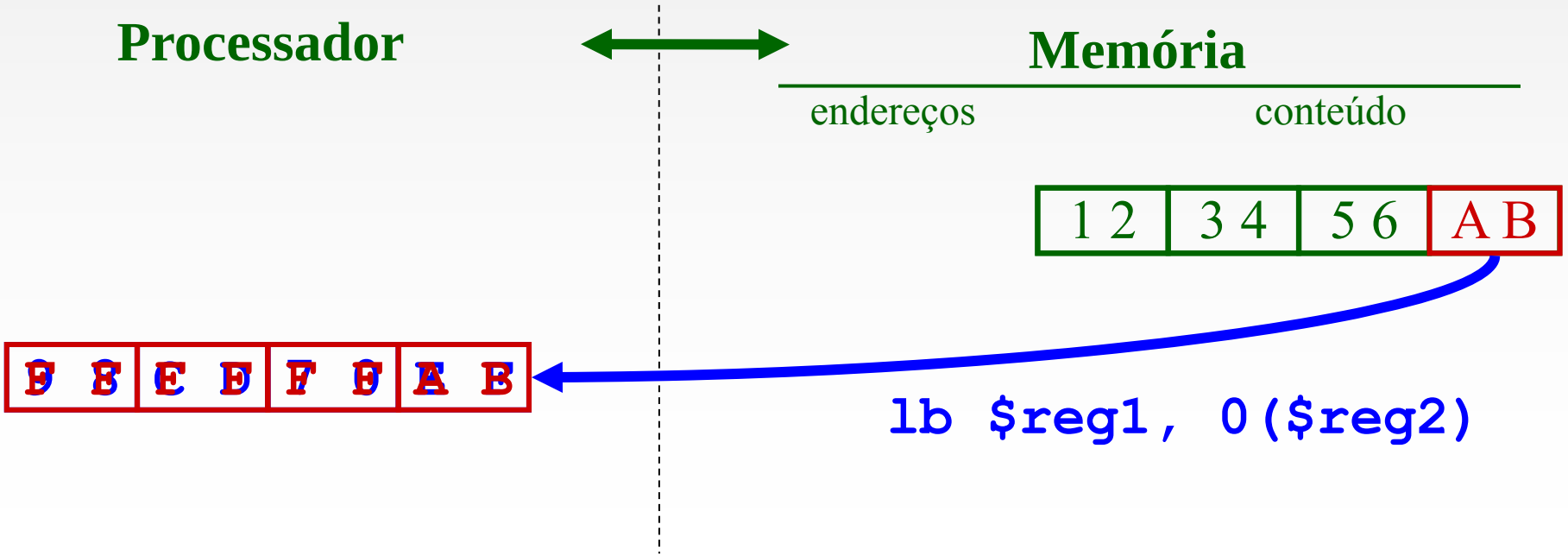
- Transferência de Dados
- Lógicas
- Controle
- Suporte a procedimentos

Instruções MIPS
:: Transferência de dados

- MIPS oferece instruções para mover bytes, halfwords e doublewords:
  - Load byte: **`lb`**
  - Store byte: **`sb`**
  - Load halfword: **`lh`**
  - Store halfword: **`sh`**
  - Load doubleword: **`ld`**
  - Store doubleword: **`sd`**

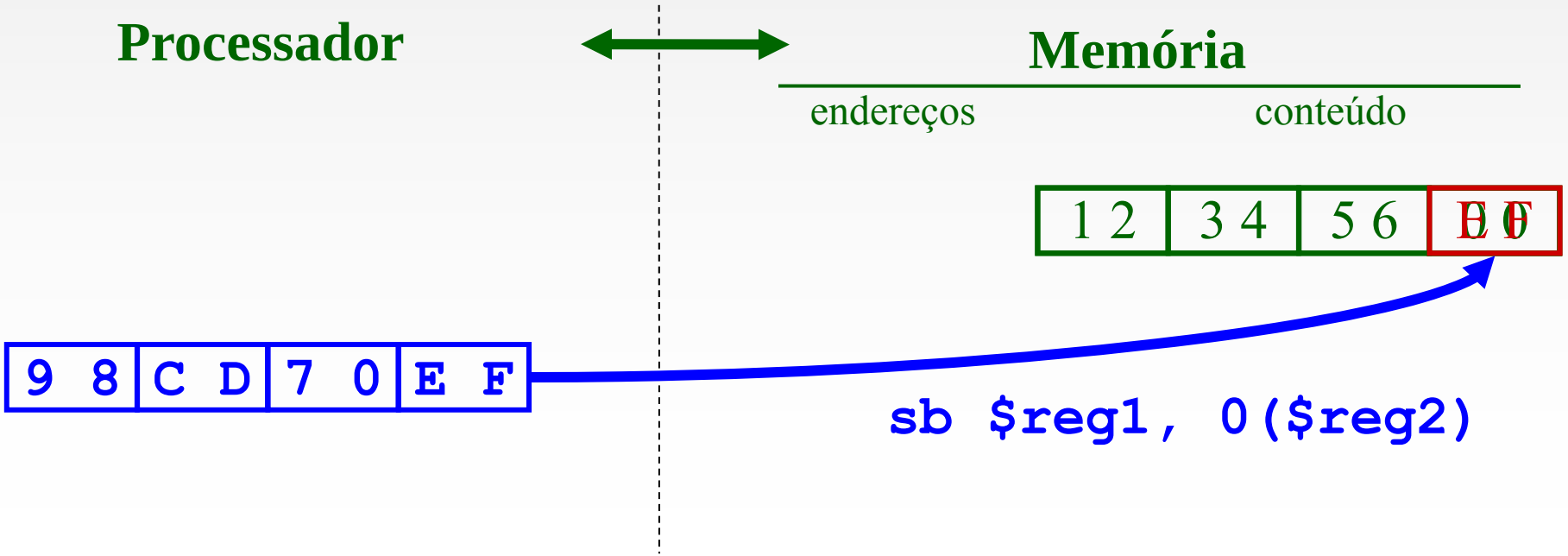
Instruções MIPS
:: Transferência de dados

- Load byte (`lb`): lê um byte da memória, colocando-o nos 8 bits mais à direita de um registrador
- Demais bits do registrador: conservam sinal do byte carregado



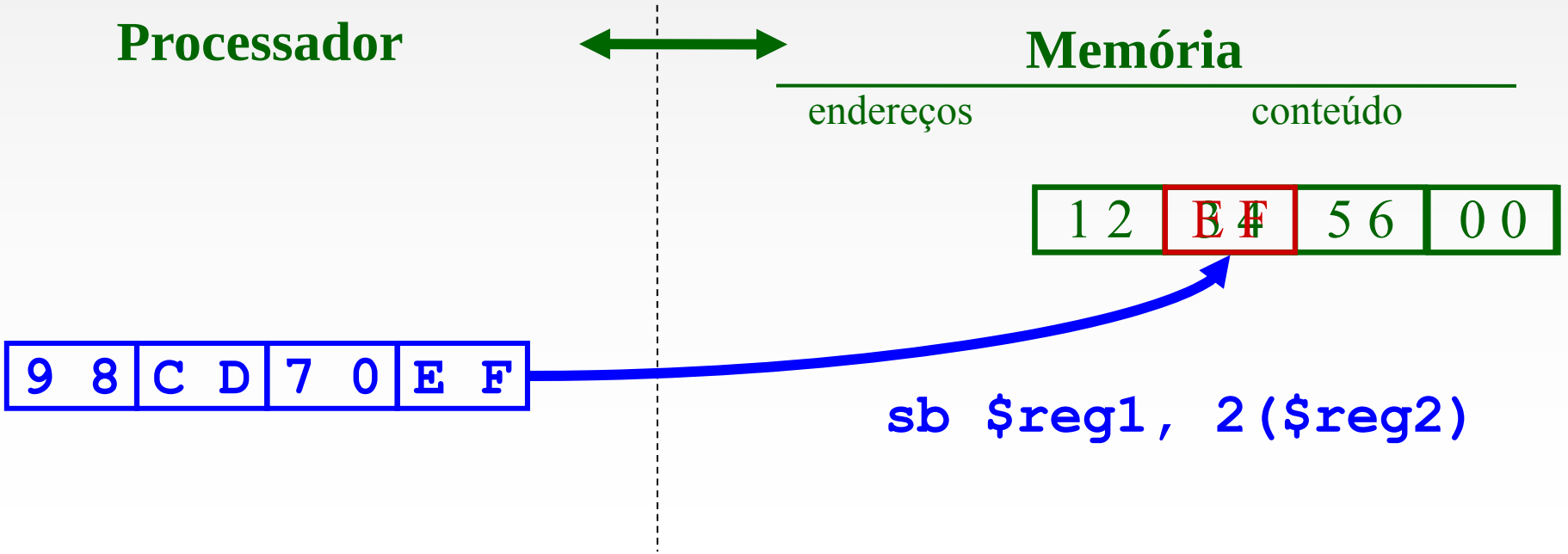
Instruções MIPS
:: Transferência de dados

- Store byte (`sb`): separa o byte mais à direita de um registrador e o escreve na memória
- Demais bits da memória: permanecem inalterados



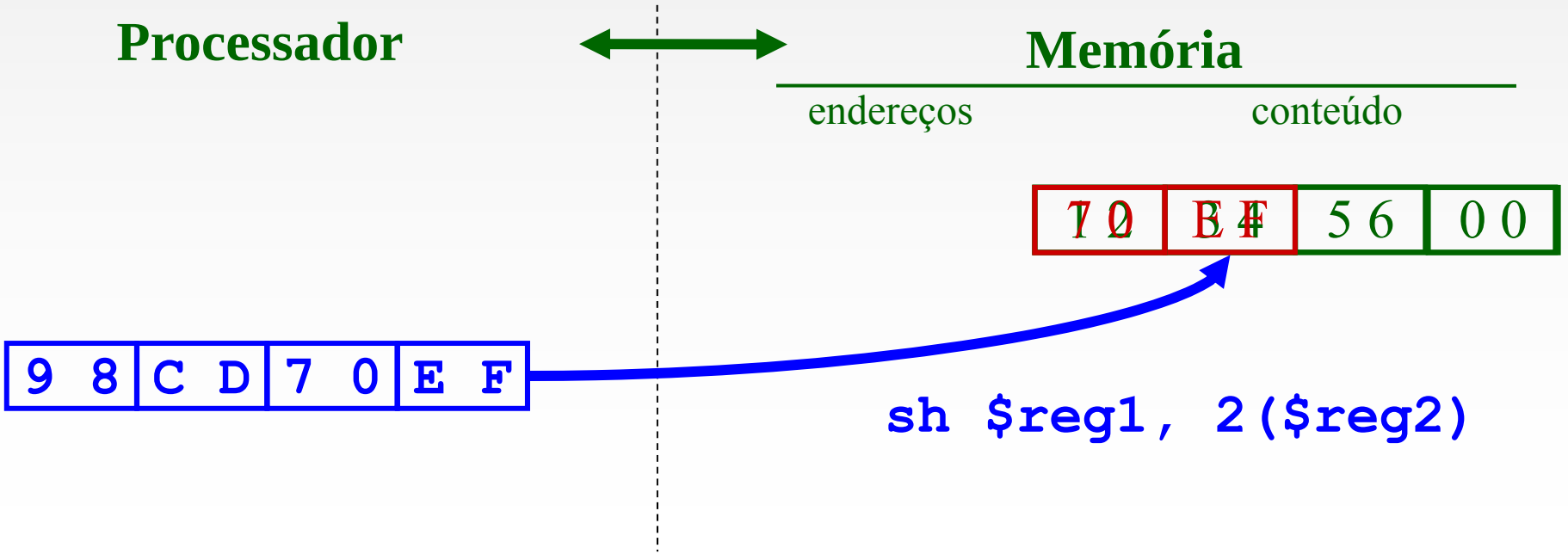
Instruções MIPS
:: Transferência de dados

- Store byte (`sb`): separa o byte mais à direita de um registrador e o escreve na memória
- Demais bits da memória: permanecem inalterados



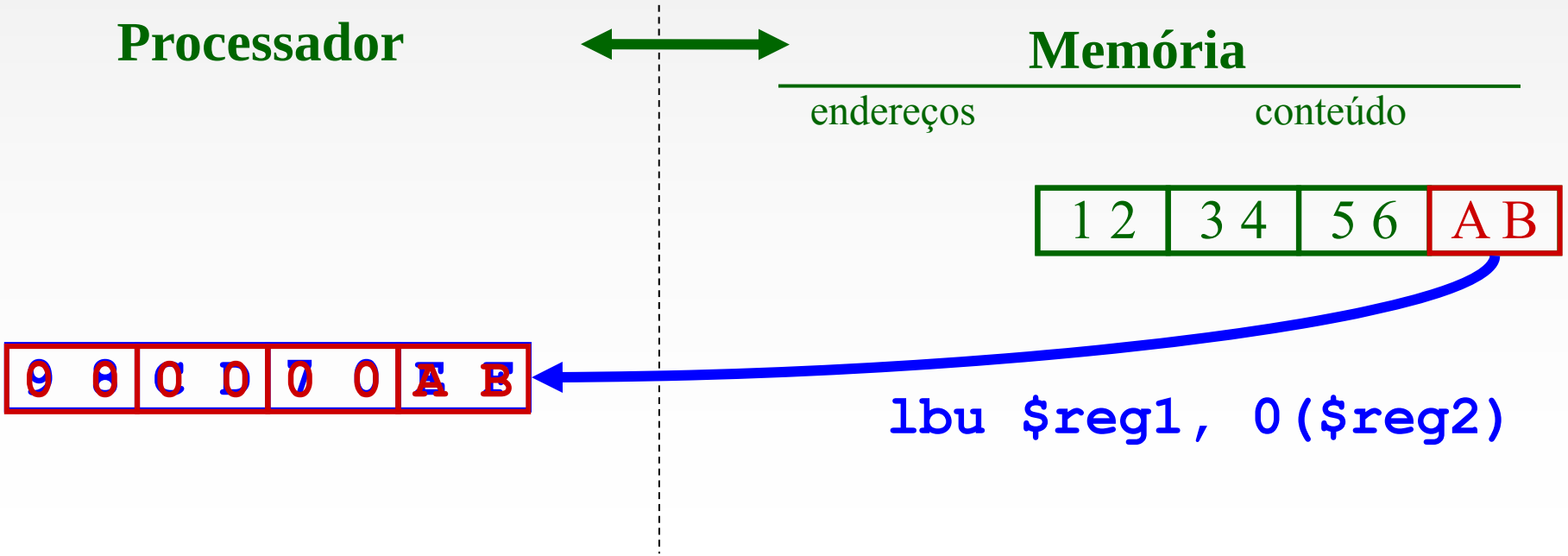
Instruções MIPS
:: Transferência de dados

- Load halfword (`lh`) e Store halfword (`sh`):
  - Mesma lógica que `lb` e `sb`, mas trabalham com halfwords (2 bytes), em vez de bytes isolados



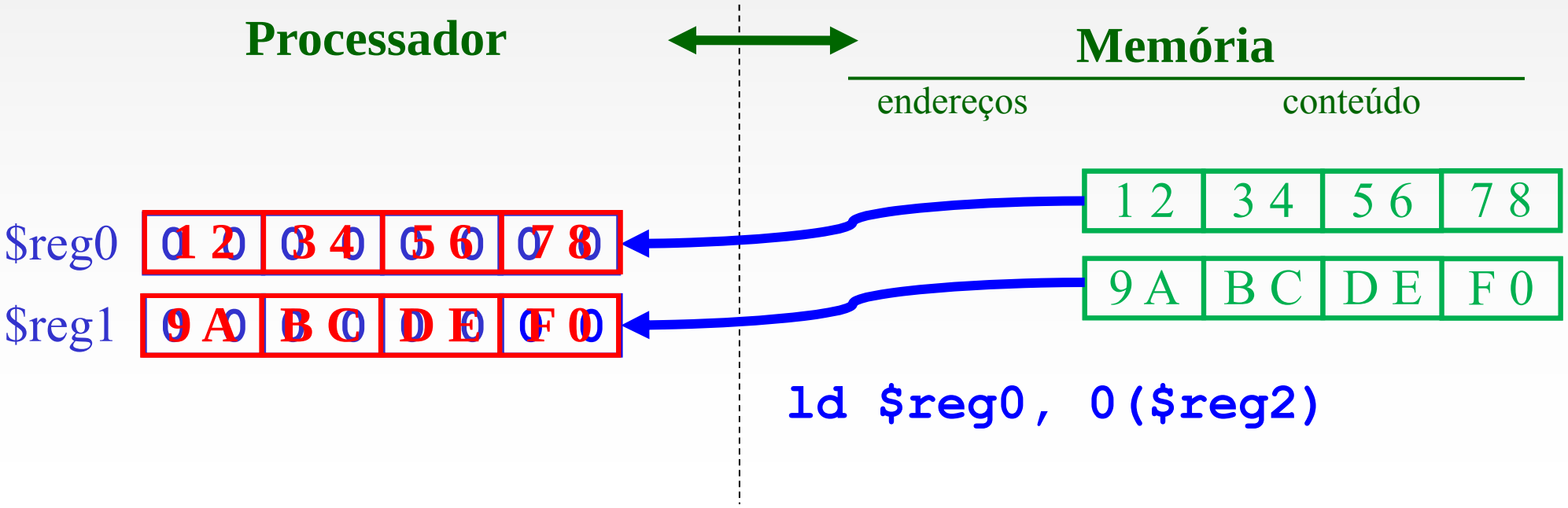
Instruções MIPS
:: Transferência de dados

- Load byte unsigned (`lbu`)/Load halfword unsigned (`lhu`)
- Lêem um byte/halfword da memória, colocando-o nos 8 bits mais à direita de um registrador
- Demais bits do registrador: preenche-se com zeros



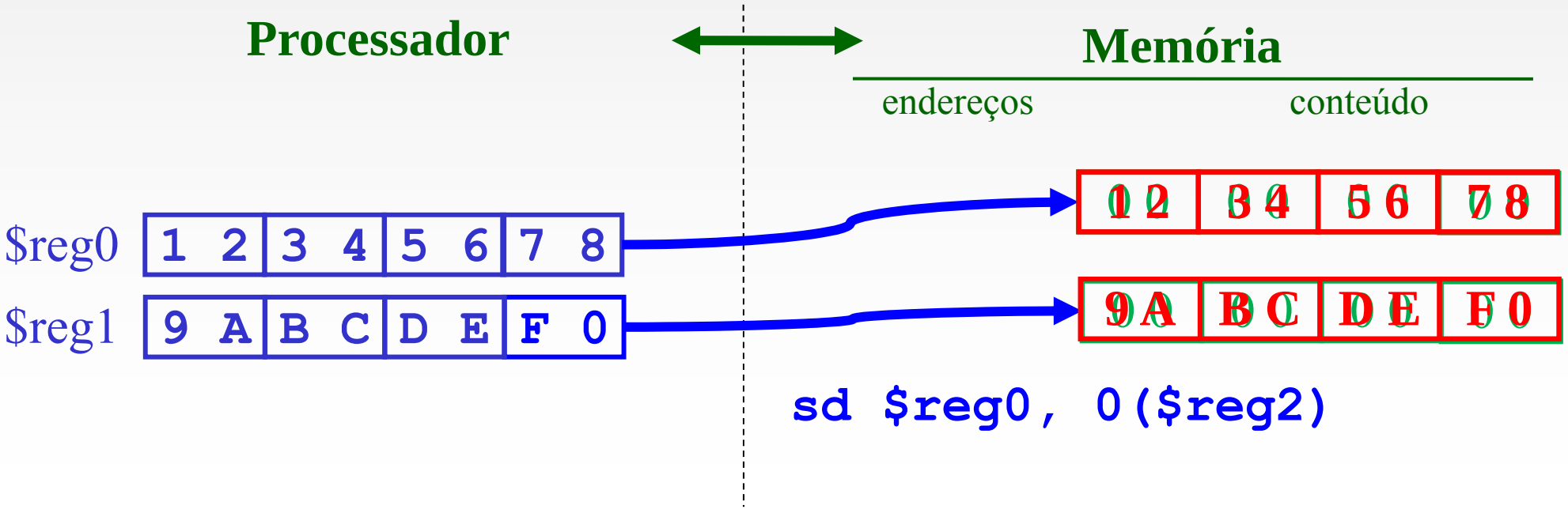
Instruções MIPS
:: Transferência de dados

- Load double word (ld)
- Le 8 bytes da memória, colocadando-os no registrador e no registrador seguinte



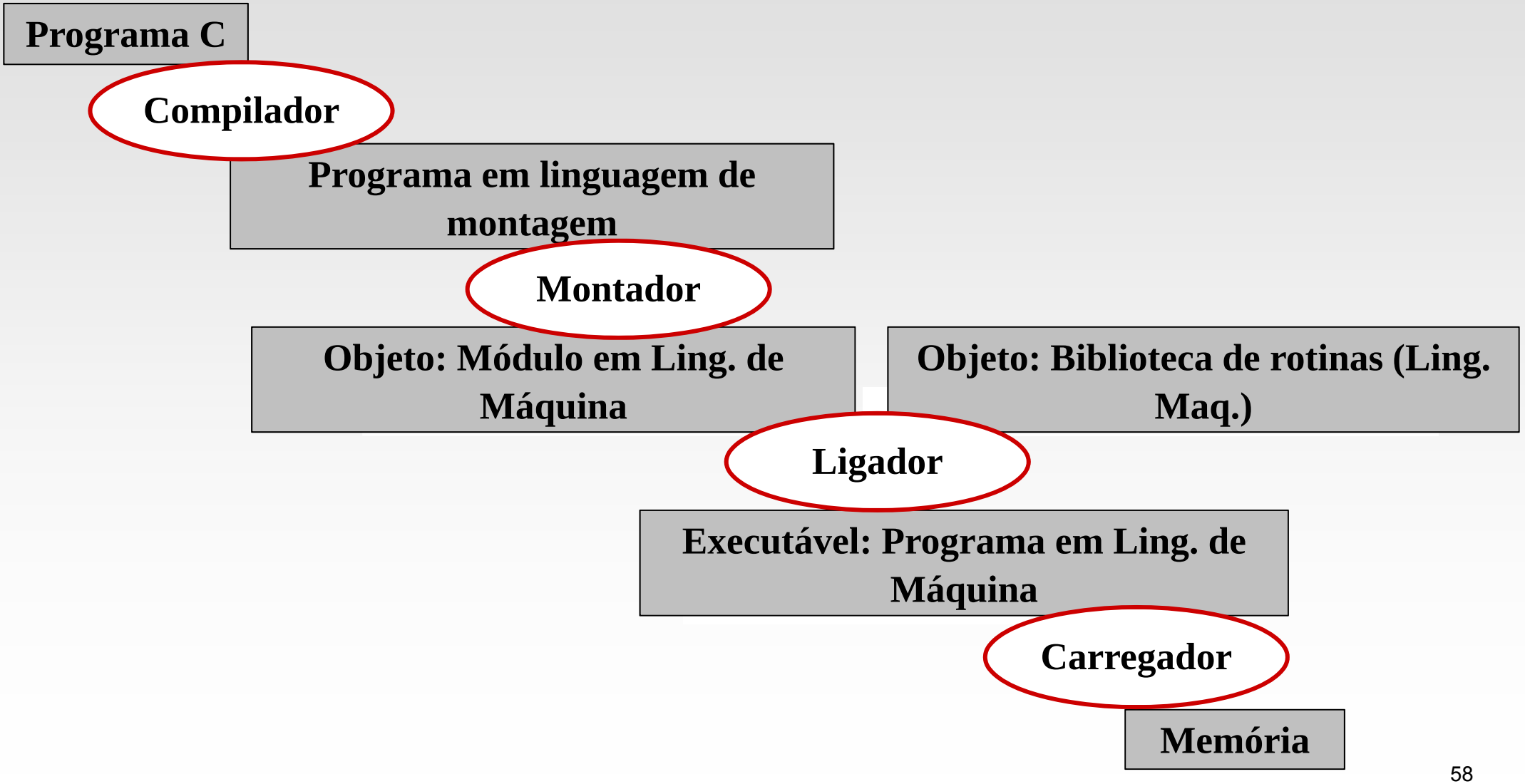
Instruções MIPS
:: Transferência de dados

- Store double word (sd)
- Escreve dois words de um registrador e do registrador seguinte na memória



Pseudo-Instruções

Hierarquia de Tradução



Pseudo-instruções

- São instruções fornecidas por um montador mas não implementadas pelo hardware MIPS
- O montador as aceita como instruções comuns, mas as traduzem para instruções equivalentes em linguagem de máquina
- Facilitam o trabalho do programador por oferecer um conjunto mais rico de instruções que aquele implementado em hardware, sem complicar este

Pseudo-instruções

- O montador utiliza o registrador `$at` para traduzir as pseudo-instruções em linguagem de máquina

- Ao se considerar o desempenho (número de instruções por programa – N_{instr}), deve-se contar as instruções reais do MIPS

Pseudo-instruções

- MAL (MIPS Assembly Language): conjunto de instruções que o programador pode utilizar para escrever um código assembly MIPS, o que inclui pseudo-instruções

- TAL (True Assembly Language): conjunto de instruções que podem realmente ser traduzidas em instruções de linguagem de máquina (strings de 32 bits binários)

Pseudo-instruções :: Exemplos

- Instruções de transferência de dados:
  - **Carregar endereço (load address)**

```
la   Rdest, Label
```

- Exemplos:

```
la     $t2, label
```

```
lui  $at, upper 16 bits
ori  $t2, $at, lower 16 bits
```

Pseudo-instruções :: Exemplos

- Instruções de transferência de dados:
  - **Carregar imediato (load immediate)**

```
li   Rdest, Const
```

- Exemplos:

```
li   $t2, const
```

```
lui  $at, upper 16 bits
ori  $t2, $at, lower 16 bits
```

Pseudo-instruções :: Exemplos

- Instruções de transferência de dados:
  - **Mover (move)**

```
move   Rdest, Rsrc
```

- Exemplos:

```
move   $t2, $t1
```

```
addu $t2, $zero, $t1
```

Pseudo-instruções :: Exemplos

- Instruções de rotação:

```
rol     Rdest, Rsrc1, Shamt
ror     Rdest, Rsrc1, Shamt
```

- Exemplo:

```
ror     $t2, $t2, 31
```

```
sll     $at, $t2, 1
srl     $t2, $t2, 31
or      $t2, $t2, $at
```

Pseudo-instruções :: Exemplos

- Instruções de desvio:
  - **Desviar para endereço relativo de 16-bits (branch)**

```
b    label
```

- Exemplo:

```
b    target
```

```
bgez $0, target *
```

**bgez - Branch if greater or equal zero*

Pseudo-instruções :: Exemplos

- Instruções de desvio:
  - **Desviar se Rsrc1 > Rsrc2**

```
bgt    Rsrc1, Rsrc2
```

  - **Desviar se Rsrc1 ≥ Rsrc2**

```
bge    Rsrc1, Rsrc2
```

  - **Desviar se Rsrc1 < Rsrc2**

```
blt    Rsrc1, Rsrc2
```

  - **Desviar se Rsrc1 ≤ Rsrc2**

```
ble    Rsrc1, Rsrc2
```

Pseudo-instruções :: Exemplos

- Pseudo-código

Branch if Greater than

```
bgt Rs, Rt, Label
```

- Instrução MIPS

```
slt $at, Rt, Rs
bne $at, $0, Label
```

Mais exemplos: Apêndice A do livro do Patterson

Chamadas de Sistema

Chamadas de sistema

- A instrução **`syscall`** suspende a execução do programa usuário e transfere o controle para o sistema operacional

- O sistema operacional acessa então o conteúdo do registrador **`$v0`** para determinar qual tarefa o programa usuário está solicitando para ser realizada

- Quando o sistema operacional termina de cumprir sua tarefa, o controle retorna ao programa usuário, onde a próxima instrução é executada

Chamadas de sistema

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_sting	4	$a0 = string address	
read_int	5		Integer in $v0
read_float	6		Float in $f0
read_double	7		Double in $f0
read_string	8	$a0 = buffer address $a1 = buffer size	
sbrk	9		Address in $v0
exit	10		

Chamadas de sistema
:: Exemplo de leitura de inteiros

```
## Get first number from user, put into $t0

li   $v0, 5     # load syscall read_int into $v0
syscall         # make the syscall
move $t0, $v0   # move the number read into $t0
```

Chamadas de sistema
:: Exemplo de escrita de inteiros

```
## Print out $t2

move $a0, $t2  # move the number to print into $a0
li   $v0, 1    # load syscall print_int into $v0
syscall        # make the syscall.
```

Chamadas de sistema
:: Exemplo de saída do programa

```
li $v0, 10   # syscall code 10 is for exit.
syscall      # make the syscall.
```

O que vocês aprenderam hoje?

- Suporte a Procedimentos
  - Realização dos procedimentos com MIPS
  - Use de registradores
  - Pilha
- Transferência de dados avançada
- Pseudo-Instruções
- Chamadas de Sistema

Assuntos da prova

- 1ª prova na próxima aula com assuntos seguintes:
  - Todos os assuntos das primeiras 4 aulas (livro: capítulos 1 e 2)
  - Organização de um computador
  - Ciclo de instruções
  - Características da arquitetura e das instruções do MIPS (mas **não** o nome e o *opcode* da cada instrução)
  - Conversão para *assembly* e código de máquina
  - Procedimentos
  - ...

Questões

Converta o código em linguagem de alto nível para o código assembly correspondente!

-- Suponha que os valores das variáveis

-- x e y do main() estejam armazenados em $s1 e $s2

```
main () {
        int x, y;
        x = 2;
        y = funcao (x);
}
int funcao (a) {
        int x;
        x = 2 * a;
        return(x);
}
```

```
funcao:   sll $v0,$a0,1
          jr $ra

main:     addi $s1, $zero,2
          add $a0, $zero, $s1
          jal funcao
          add $s2, $v0, $zero
```

Diretivas

Diretivas

- Permitem estabelecer algumas estruturas de dados iniciais que serão acessadas pelo computador durante a execução.

- As diretivas assembler começam com um ponto:

```
.data
.space
.text
```

etc.

Diretivas

- Não são executadas pelo computador durante o tempo de execução

- Apenas direcionam o montador a reservar algumas estruturas de dados antes da execução

Diretivas
:: Exemplo (1)

```
main:
        .data                   # put things into the data segment
hello_msg:
        .asciiz "Hello World\n"
        .text                   # put things into the text segment
        la $a0, hello_msg       # load the addr of hello_msg into $a0.
        li $v0, 4               # 4 is the print_string syscall.
        syscall                 # do the syscall.

        li $v0, 10              # 10 is the exit syscall.
        syscall                 # do the syscall.
```

Diretivas
:: Exemplo (2)

Código C:

```
int MATRIZ[1024];
```

Código assembly MIPS:

```
        .data           # dados devem ser alocados
                        # no segmento de dados
MATRIZ: .space 4096
```

Diretivas
:: Exemplo (3)

Código C:

```
int Pof2[16] = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512,
                1024, 2048,4096, 8192, 16384, 32768}
```

Código assembly MIPS:

```
      .data
Pof2: .word 1, 2, 4, 8, 16, 32, 64, 128, 256, 512,
            1024, 2048, 4096, 8192, 16384, 32768
```

Diretivas
:: Carregar endereço base

- Pseudo-instrução load address (`la`)

```
        .data
pof2:  .word 1, 2, 4, 8, 16, 32, 64, 128, 256,
             512, 1024, 2048, 4096, 8192, 16384

      la $t0, pof2
```

- Carrega no registrador `$t0` o endereço de memória onde o matriz `pof2` começa

Diretivas
:: Elementos do array

- Instrução load word, halfword, byte (`lw`, `lh`, `lb`)
- Depende do tipo de diretiva declarada
- Offset de endereço de memória é sempre dado em bytes!

Diretivas

Name	Parameters	Description
`.align`	*n*	Align the next item on the next 2^n-byte boundary. `.align 0` turns off automatic alignment.
`.ascii`	*str*	Assemble the given string in memory. Do not null-terminate.
`.asciiz`	*str*	Assemble the given string in memory. Do null-terminate.
`.byte`	*byte1* $\cdots$ *byteN*	Assemble the given bytes (8-bit integers).
`.half`	*half1* $\cdots$ *halfN*	Assemble the given halfwords (16-bit integers).
`.space`	*size*	Allocate *n* bytes of space in the current segment. In SPIM, this is only permitted in the data segment.
`.word`	*word1* $\cdots$ *wordN*	Assemble the given words (32-bit integers).