

“Se não existisse C hoje estaríamos  
programando em Obol, Pasa! e BASI!”

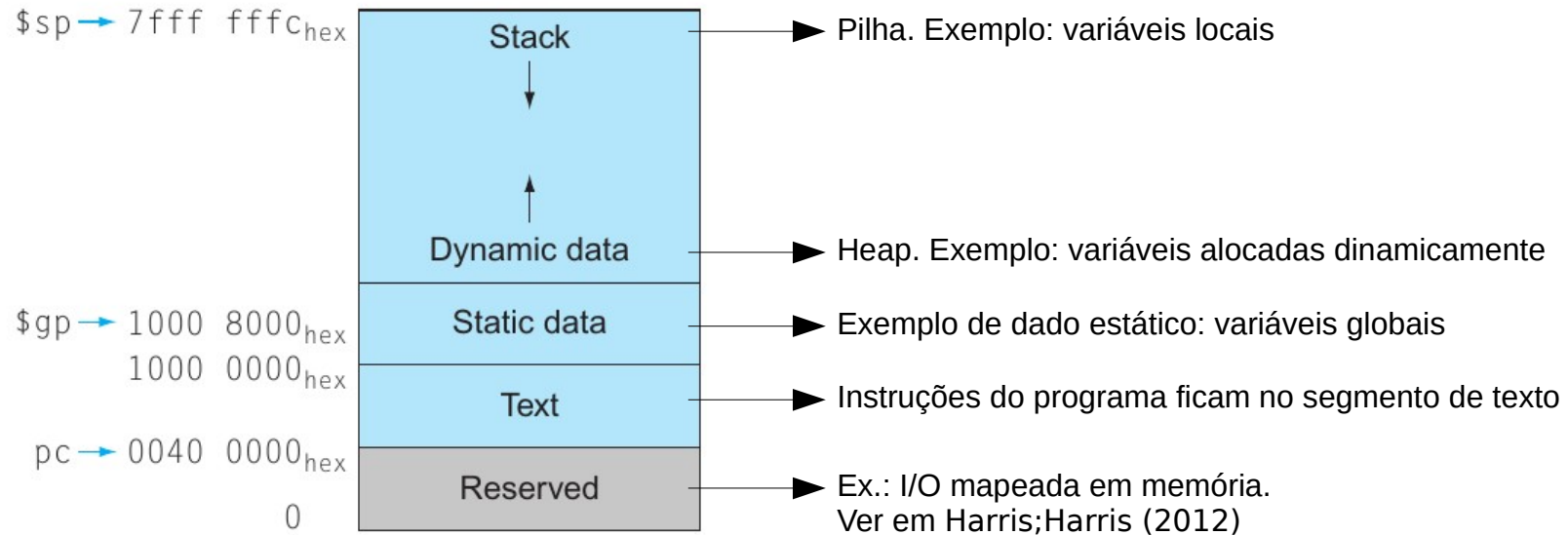
# Conjuntos de Instruções

Contador de Programa, Branches  
e Jumps

Paulo Ricardo Lisboa de Almeida

# Convenção da memória

- A imagem a seguir é uma convenção sobre como um programa fica na memória principal da máquina (ex.: Memória RAM)
- Por ser uma convenção, e não uma especificação, pode mudar de implementação para implementação

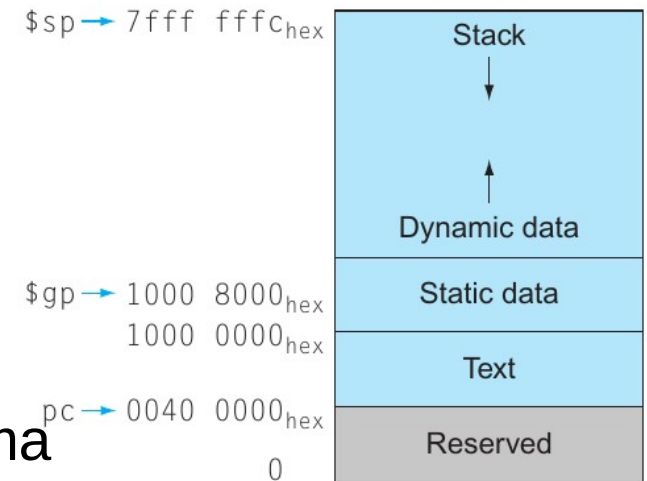


# Contador de Programa

- Considere o seguinte programa em assembly do MIPS

ENDEREÇO(hexa)	INSTRUÇÃO
00400000	ori \$t2,\$zero, 25
00400004	lw \$t3, 0(\$s0)
00400008	add \$t4, \$t2, \$t3
0040000C	sub \$t5, \$t2, \$t3

- O processador sabe qual a próxima instrução a ser executada através do contador de programa
  - Registrador **PC** (Program Counter)
    - No x86 o PC é chamado de IP (Instruction Pointer)
  - Não é diretamente visível/acessível** ao programador



# Contador de Programa

- Durante a execução
  - O processador carrega a instrução no endereço apontado pelo registrador PC
  - Antes mesmo de executar a instrução carregada, o processador acrescenta +4 no PC para apontar para próxima instrução
  - O Processador executa a instrução carregada
  - O processo se repete

## ENDEREÇO(hexa)

00400000  
00400004  
00400008  
0040000C

## INSTRUÇÃO

ori \$t2, \$zero, 25  
lw \$t3, 0(\$s0)  
add \$t4, \$t2, \$t3  
sub \$t5, \$t2, \$t3

# Exemplo

pc = 0x00400000

ENDEREÇO(hexa)	INSTRUÇÃO
00400000	ori \$t2, \$zero, 25
00400004	lw \$t3, 0(\$s0)
00400008	add \$t4, \$t2, \$t3
0040000C	sub \$t5, \$t2, \$t3

# Exemplo

pc = 0x00400004

ENDEREÇO(hexa)	INSTRUÇÃO
00400000	ori \$t2, \$zero, 25
00400004	lw \$t3, 0(\$s0)
00400008	add \$t4, \$t2, \$t3
0040000C	sub \$t5, \$t2, \$t3

# Exemplo

pc = 0x00400008

ENDEREÇO(hexa)	INSTRUÇÃO
00400000	ori \$t2, \$zero, 25
00400004	lw \$t3, 0(\$s0)
00400008	add \$t4, \$t2, \$t3
0040000C	sub \$t5, \$t2, \$t3

# Exemplo

pc = 0x0040000C

ENDEREÇO(hexa)	INSTRUÇÃO
00400000	ori \$t2, \$zero, 25
00400004	lw \$t3, 0(\$s0)
00400008	add \$t4, \$t2, \$t3
0040000C	sub \$t5, \$t2, \$t3



# Contador de Programa

- Como os demais registradores, o contador de programa armazena 32 bits
  - **Pergunta: Qual o maior programa que podemos escrever em uma arquitetura MIPS de 32 bits?**

# Contador de Programa

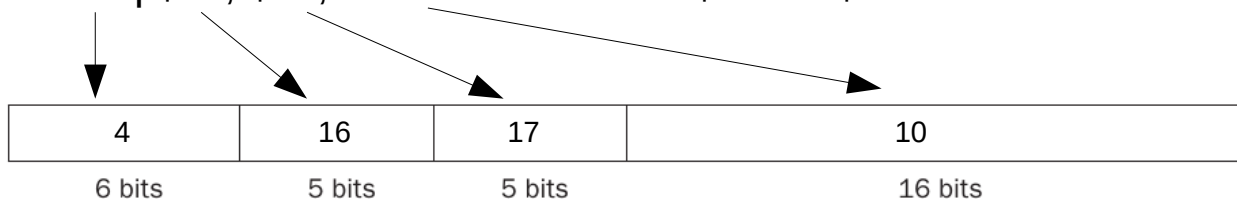
- Como os demais registradores, o contador de programa armazena 32 bits
  - **Pergunta: Qual o maior programa que podemos escrever em uma arquitetura MIPS de 32 bits?**
  - $2^{32} = 4\text{GiB}$
  - Na prática esse número é muito menor, já que o programa não é composto somente de instruções
    - Temos o segmento de pilha, heap, dados estáticos, ...
  - **Pergunta: Considerando que a primeira instrução do programa está em  $00400000_{16}$ , existe a possibilidade de em algum momento pc conter, por exemplo, o valor  $00400003_{16}$  no MIPS32?**

# Contador de Programa

- Pergunta: Considerando que a primeira instrução do programa está em 0040000016, existe a possibilidade de em algum momento pc conter, por exemplo, o valor 0040000316 no MIPS32?
  - Não. Toda instrução ocupa 32 bits (4 bytes). Como a memória é endereçada em bytes, os saltos são de 4 em 4
  - As instruções sempre começam em um endereço múltiplo de 4
    - **Restrição de alinhamento**
  - Comum em muitas arquiteturas
  - Não existe essa restrição em x86

# Branches

- Branch → desvio
- Desvio condicionais
  - Instruções utilizadas para tomada de decisão
  - Construir os “ifs” e “loops”
- beq ← branch if equal (desvie se igual)
  - Formato:
  - beq reg1, reg2, ENDEREÇO #salte ao ENDEREÇO se reg1 == reg2
- Instrução do tipo-I
- Exemplo concreto:
  - beq \$s0, \$s1, 10 #salte 10 se \$s0 == \$s1



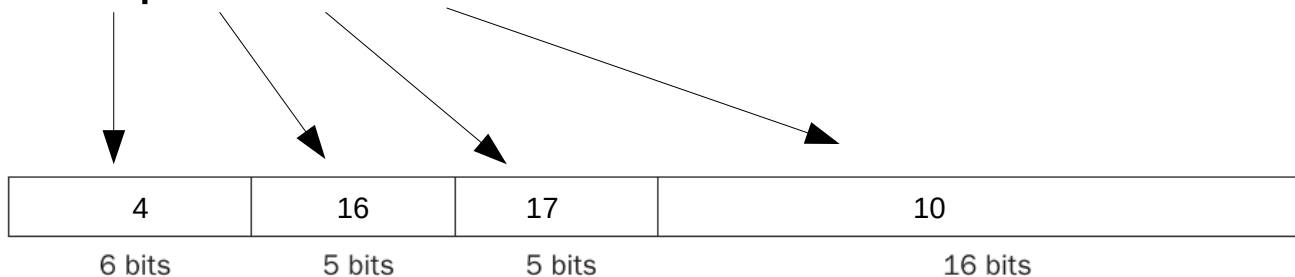
Valores em decimal

# Branches

- `bne` ← branch if not equal (desvie se não igual)
- Formato:  
    `bne reg1, reg2, ENDEREÇO #salte ao endereço se reg1 != reg2`
- Exemplo concreto:  
    `bne $s0, $s1, 10 #salte 10 se $s0 != $s1`

# Branches

beq \$s0, \$s1, 10      #salte 10 se \$s0 == \$s1



- Problema
  - Se o ENDEREÇO apontar para o endereço real da instrução, nenhum programa poderia conter mais de  $2^{16} = 64\text{Kbytes}$  de instruções
  - Muito pequeno
  - **Solução?**

# Branches

- Problema
  - Se o ENDEREÇO apontar para o endereço real da instrução, nenhum programa poderia conter mais de  $2^{16} = 64\text{Kbytes}$  de instruções
  - Solução?
    - Os desvios geralmente são tomados **para regiões próximas da instrução atual**
    - O registrador PC aponta para a próxima instrução a ser executada
    - Dessa forma, o ENDEREÇO em um branch é um “salto” referente ao pc
      - Para aumentar o alcance, o salto é definido em palavras (4 bytes)
      - O salto pode ser positivo ou negativo
        - Alcance de  $\pm 2^{15}$
      - **Endereçamento relativo ao PC**
    - Logo, caso a condição do branch se satisfaça o endereço efetivo do salto é
$$pc = pc + 4 + ENDERECO * 4$$

# Exercício

Endereço(Hexa)	Instrução
00400000	lw \$s0, 0(\$t0)
00400004	lw \$s1, 4(\$t0)
00400008	lw \$s2, 8(\$t0)
0040000C	beq \$s0,\$s1,ENDEREÇO???
00400010	addi \$s2, \$s2, 5
00400014	addi \$s2, \$s2, 10

← Desejamos ignorar essa instrução se \$s0 == \$s1

- Qual o valor devemos colocar em “ENDEREÇO???”, considerando que caso \$s0 seja igual a \$s1, devemos saltar para a instrução 00400014<sub>16</sub>?



# Exercício

Endereço(Hexa)	Instrução
00400000	lw \$s0, 0(\$t0)
00400004	lw \$s1, 4(\$t0)
00400008	lw \$s2, 8(\$t0)
0040000C	beq \$s0,\$s1,ENDEREÇO???
00400010	addi \$s2, \$s2, 5
00400014	addi \$s2, \$s2, 10

← Desejamos ignorar essa instrução se \$s0 == \$s1

- Qual o valor devemos colocar em “ENDEREÇO???”?
  - Ao chegar na instrução beq,  $pc = 0040000C_{16}$ 
    - A instrução é carregada para a CPU, e **antes de executar a instrução**, pc é incrementado e aponta para a próxima, ou seja,  $00400010_{16}$ 
      - Então o beq deve assumir que o salto deve ser feito a partir de  $00400010_{16}$
    - Como desejamos saltar para  $00400014_{16}$ ,  $00400014_{16} - 00400010_{16} = 4_{16}$
    - Como **o salto é feito em palavras**,  $4_{16} / 4_{16} = 1_{16}$  palavra
    - Logo endereço deve conter  $1_{16} = 1_{10}$

# Assembler ao resgate

- Lidar com os endereços dos branches não é tarefa simples
  - Calcular o endereço pode ser confuso
  - Ao inserir uma instrução entre o branch e o seu endereço final, temos que atualizar o endereço
- O **montador** nos poupa desse problema
- Podemos utilizar **rótulos** (labels) no programa, e pedir por um desvio para o rótulo
  - O montador se encarrega de substituir o rótulo pelo endereço correto **quando o programa for montado**
  - **Rótulos são definidos com um nome único, seguido de dois pontos**

# Utilização de rótulos. Exemplo

- Exemplo

```
lw $s0, 0($t0)
lw $s1, 4($t0)
lw $s2, 8($t0)
beq $s0,$s1,salto
addi $s2, $s2, 5
```

**salto:**

```
addi $s2, 10
```

# Comparações

- `slt` ← set on less than (atribuir se menor que)
  - Instrução do tipo-R  
`slt $regResultado, $reg1, $reg2`
    - `$regResultado = 1` se `$reg1 < $reg2`, ou recebe 0 caso contrário
- Exemplo concreto  
`slt $t0, $s3, $s4` # `$t0` recebe 1 se `$s3 < $s4`, ou 0 caso contrário
- Variantes
  - `slti` para imediatos
  - `sltu` para comparações sem sinal
  - `sltiu` para comparações imediatas sem sinal

# Exercício

- Considere o seguinte trecho de código em C

```
if(a > b){  
    a += 30;  
}  
b += 10;
```
- Assumindo que a variável *a* está no registrador \$s0, e *b* no registrador \$s1, como fica esse trecho em Assembly do MIPS?

# Exercício

- Considere o seguinte trecho de código em C

```
if(a > b){  
    a += 30;  
}  
b += 10;
```
- Assumindo que a variável *a* está no registrador \$s0, e *b* no registrador \$s1, como fica esse trecho em Assembly do MIPS?

```
slt $t0,$s1,$s0  
beq $t0, $zero, b_maior_igual  
addi $s0,$s0,30  
b_maior_igual:  
addi $s1,$s1,10
```

# Saltos incondicionais

- $j \leftarrow \text{jump}$ 
  - Salte para o endereço

- Formato

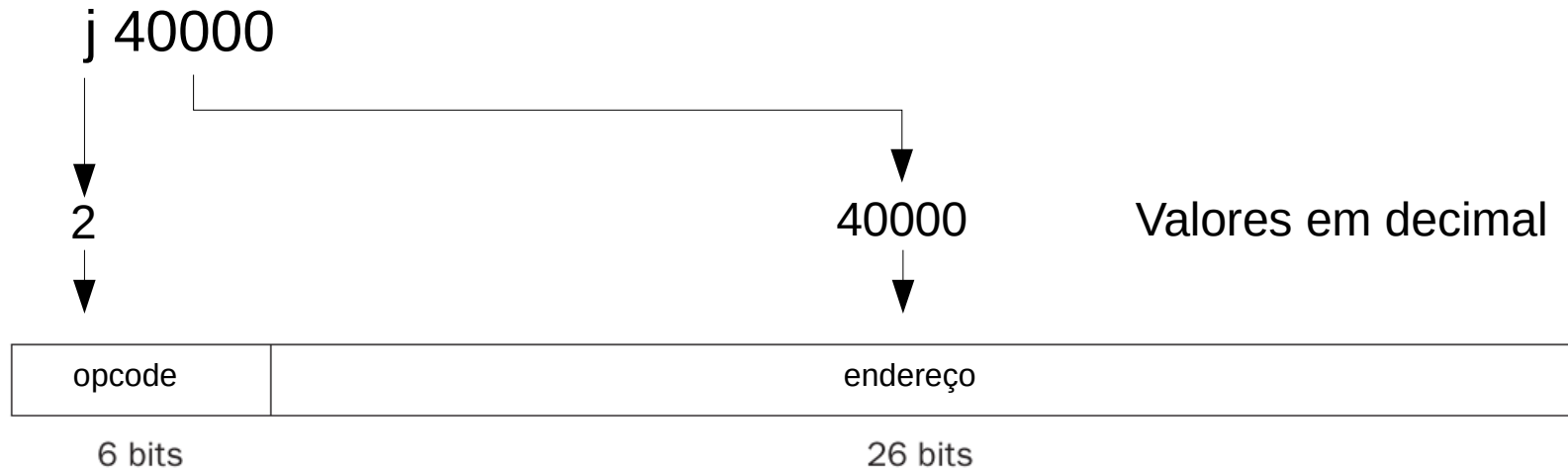
$j$  ENDEREÇO

- Exemplo concreto

$j$  40000    #salta para a palavra 40000

# Saltos incondicionais

- Jumps são instruções do **Tipo-j**
- Tipo mais simples de instrução





# Saltos incondicionais

- Diferente dos desvios condicionais, os jumps **não são relativos ao pc**
  - Estamos efetivamente saltando para a palavra 40000 no exemplo anterior
- Como o **endereçamento é em palavras** mais uma vez, multiplicamos por 4 para obter o endereço em bytes da instrução
  - Ou seja,  $pc = \text{ENDEREÇO} \ll 2$  #deslocado 2 bits para multiplicar por 4
- Com isso, o salto tem capacidade efetiva de atingir endereços de até 28 bits
- Os 4 bits mais altos de pc ainda são emprestados para completar os 32 bits necessários para representar um endereço completo
  - Endereçamento **pseudodireto**
- Da mesma forma que com branches, **podemos utilizar rótulos, e deixar o cálculo do endereço efetivo a cargo do montador**

# Exercício

- Considere o seguinte trecho de código em C

```
while(vet[i] == k){  
    i +=1;  
}  
vet[i] = k+10;
```
- Assumindo que as variáveis  $i$  e  $k$  se encontram nos registradores \$s3 e \$s5, e que a base do vetor *vet* está em \$s6, como fica o trecho em assembly do MIPS? Considere ainda que o vetor é de inteiros, e que cada inteiro ocupa uma palavra.

# Exercício - Solução

loop:

sll \$t0,\$s3,2	#multiplicando i por 4 para ajustar as palavras
add \$t0,\$t0,\$s6	#adicionando o deslocamento à base do vetor
lw \$t1,0(\$t0)	#\$t1 = vet[i]
bne \$t1,\$s5,saida	#saia se vet[i] != k
add \$s3, \$s3, 1	#adicionando 1 em i (corpo do loop)
j loop	#depois de executar o corpo, retorna para o início

saida:

addi \$t1, \$s5, 10	#\$t1 = k+10
sw \$t1,0(\$t0)	#vet[i] = \$t1, ou seja, k+10

# Exercício

- Considere o seguinte desvio

```
beq $s0, $s1, L1  
#conjunto de instruções 1
```

```
L1:  
#conjunto de instruções 2
```

- Considere que o número de instruções entre o beq e L1 é muito grande, e não pode ser endereçado no campo de 16 bits do beq (instrução do Tipo-I). Como resolver esse problema adicionando um jump extra?

# Exercício - Solução

```
bne $s0, $s1, L2
```

```
j L1
```

L2:

```
#conjunto de instruções 1
```

L1:

```
#conjunto de instruções 2
```

Com essa solução podemos fazer saltos de até 256MB! Saltos maiores são possíveis com instruções jump register, que veremos nas próximas aulas, ou adicionando-se jumps extras no meio do código.

# Exercício

- Considere o programa em C a seguir,

```
if((a<b && b < 50) || a == -10){  
    vet[b] = vet[b] + vet[b-20];  
}else{  
    a = 50;  
}  
b++;
```
- Assumindo que as variáveis *a* e *b* estão nos registradores \$s0 e \$s1, respectivamente, e que o endereço base de *vet* está em \$s2. Considerando também que o vetor é de inteiros, e que cada inteiro ocupa uma palavra, escreva o programa equivalente em Assembly do MIPS.

# Exercício – Possível resposta

```
    slt $t0,$s0,$s1    # $t0 = 1 se a < b
    beq $t0, $zero, L1 # se a era maior ou igual a B, $t0=0 e então pula para o teste se a == -10
    slti $t0,$s1,50    # faz a próxima verificação. $t0 = 1 se b < 50
    beq $t0, $zero, L1 # se b era maior ou igual a 50, então pula para o teste se a == -10
    j if               # passou pelas duas primeiras condições, então pula para o if
L1:
    ori $t0, -10       # carrega a constante -10 para $t0
    bne $s0,$t0,else   # se a != -10, pula para o else
if:
    sll $t0,$s1,2       # multiplicando b por 4 e salvando em $t0
    add $t0,$t0,$s2     # somando deslocamento com a base do vetor
    lw $t1,0($t0)       # $t1 = vet[b]
    addi $t2,$s1,-20    # $t2 = b - 20
    sll $t2,$t2,2       # multiplicando $t2 por 4 para obter o deslocamento em palavras
    add $t2,$t2,$s2     # somando deslocamento com a base do vetor
    lw $t2,0($t2)       # $t2 = vet[b-20]
    add $t1,$t1,$t2     # $t1 = vet[b] + vet[b+20]
    sw $t1,0($t0)       # $vet[b] = $t1
    j saida            # pula para o rótulo saída para não executar o else também
else:
    ori $s0, 50         # a=50
saida:
    addi $s1, 1         # b++
```

# Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores.** 4a Edição: Interface Hardware / Software. Elsevier Brasil, 2014.
- Andrew S. Tanenbaum. **Organização estruturada de computadores.** 5. ed. São Paulo: Pearson, 2007.
- Harris, D. and Harris, S. **Digital Design and Computer Architecture.** 2a ed. 2012.