

Infraestrutura de Hardware

Instruindo um Computador – Subrotinas, Tipos de Dados e Modos de Endereçamento

Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?
- O que determina o desempenho de um programa e como ele pode ser melhorado?
- Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?

Subrotinas

- Subrotinas são utilizadas para estruturar um programa
 - Facilita entendimento
 - Aumenta reuso de código
 - Exs: Procedimentos, funções e métodos
- Chamada de subrotina, faz com que programa execute as instruções contidas na subrotina
- Ao término da execução de uma subrotina, computador deve executar instrução seguinte à chamada de subrotina

Seis Etapas da Execução de uma Subrotina

1. Rotina que faz a chamada (**caller**) coloca argumentos em um lugar onde a subrotina chamada (**callee**) pode acessá-los
Passagem de argumentos
2. **Caller** transfere controle para o **callee**
3. **Callee** adquire os recursos de armazenamento necessários
4. **Callee** executa suas instruções
5. **Callee** (quando é o caso) coloca o valor do resultado em um lugar que **caller** pode acessá-lo
6. **Callee** retorna controle ao **caller**

Seis Etapas da Execução de uma Subrotina

1. Rotina que faz a chamada (**caller**) coloca argumentos em um lugar onde a subrotina chamada (**callee**) pode acessá-los

Passagem de argumentos

2. Caller transfere controle para o callee
3. Callee adquire os recursos de armazenamento necessários
4. Callee executa suas instruções
5. Callee (quando é o caso) coloca o valor do resultado em um lugar que caller pode acessá-lo
6. Callee retorna controle ao caller

Passagem Argumentos

Ling. alto nível

```
int media(int w, int x, int y, int z) {  
    ...  
    (corpo da função)  
    ...  
}  
/* Programa principal */  
int main() {  
    int m;  
    m = media(2,3,6,2);  
    ....  
}
```

Passando Argumentos

- No MIPS, 4 registradores são destinados para armazenar argumentos

\$a0 - \$a3 – números 4 a 7

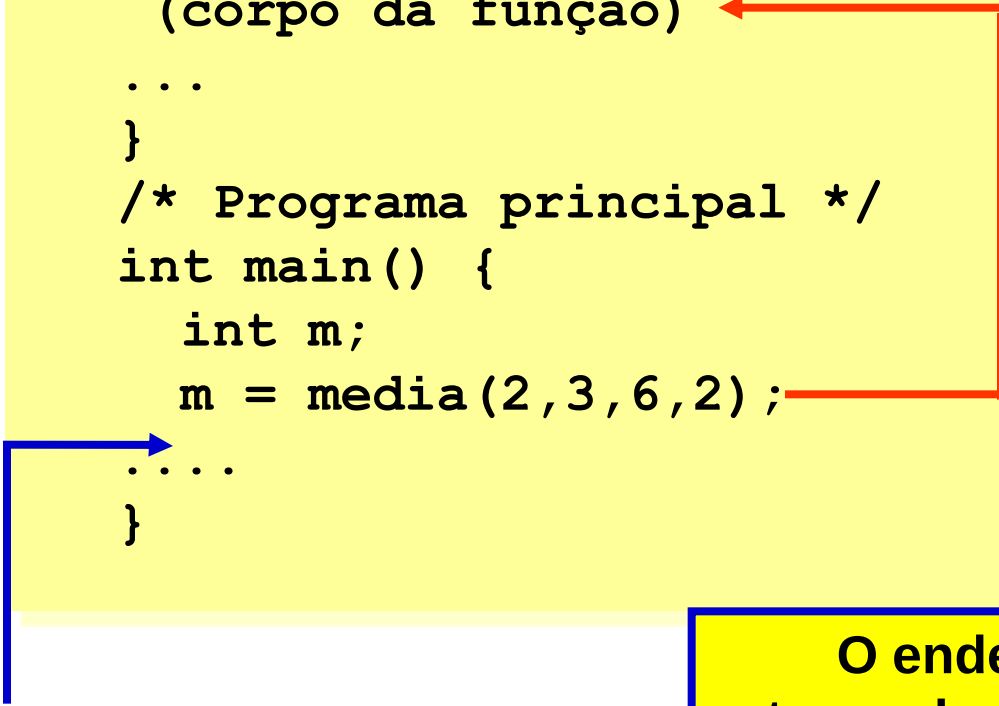
Seis Etapas da Execução de uma Subrotina

1. Rotina que faz a chamada (caller) coloca argumentos em um lugar onde a subrotina chamada (callee) pode acessá-los
Passagem de argumentos
2. **Caller** transfere controle para o **callee**
3. Callee adquire os recursos de armazenamento necessários
4. Callee executa suas instruções
5. Callee (quando é o caso) coloca o valor do resultado em um lugar que caller pode acessá-lo
6. Callee retorna controle ao caller

Tranferência de Controle Para Subrotina

Ling. alto nível

```
int media(int w, int x, int y, int z) {  
    ...  
    (corpo da função)  
    ...  
}  
/* Programa principal */  
int main() {  
    int m;  
    m = media(2,3,6,2);  
    ....  
}
```



The diagram illustrates the control flow between the `main` function and the `media` subroutine. A red arrow originates from the call `m = media(2,3,6,2);` in the `main` function and points to the `(corpo da função)` line in the `media` function, indicating the transfer of control. A blue arrow originates from the end of the `main` function (after the `....` line) and points back to the `main` function, indicating the return of control.

Executa
primeira
instrução da
subrotina

Controle deve passar
para subrotina...
mas como?

O endereço de
retorno deve ser salvo...
mas onde?

Retorno após
a chamada

Instrução Para Chamada de Subrotinas

- MIPS oferece uma instrução para fazer a chamada a subrotina

Jump And Link

- Instrução para chamar a subrotina possui um operando:

Label da subrotina

```
jal label
```

- Instrução pula para endereço inicial da subrotina e salva endereço de retorno (instrução após chamada)

\$ra – *return address* (número 31)– registrador que armazena endereço de retorno

Armazena **PC + 4**

Seis Etapas da Execução de uma Subrotina

1. Rotina que faz a chamada (caller) coloca argumentos em um lugar onde a subrotina chamada (callee) pode acessá-los
Passagem de argumentos
2. Caller transfere controle para o callee
3. **Callee** adquire os recursos de armazenamento necessários
4. **Callee** executa suas instruções
5. **Callee** (quando é o caso) coloca o valor do resultado em um lugar que **caller** pode acessá-lo
6. Callee retorna controle ao caller

Armazenamento e Retorno de Valores

Ling. alto nível

```
int media(int w, int x, int y, int z) {  
    int result;  Variável local  
    result = (w + x + y + z) / 4;  
    return result;  Retorna valor  
}  
/* Programa principal */  
int main() {  
    int m;  
    m = media(2, 3, 6, 2);  
    ....  
}
```

- Variáveis podem ser salvas em registradores disponíveis
- No MIPS, 2 registradores para valores retornados

\$v0 - \$v1 – números 2 a 3

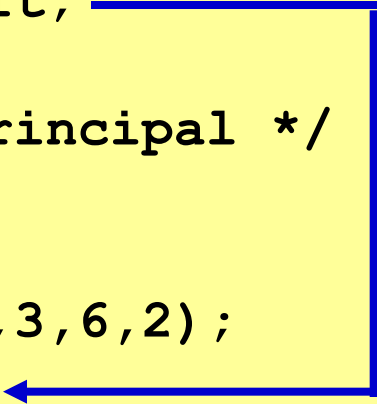
Seis Etapas da Execução de uma Subrotina

1. Rotina que faz a chamada (caller) coloca argumentos em um lugar onde a subrotina chamada (callee) pode acessá-los
Passagem de argumentos
2. Caller transfere controle para o callee
3. Callee adquire os recursos de armazenamento necessários
4. Callee executa suas instruções
5. Callee (quando é o caso) coloca o valor do resultado em um lugar que caller pode acessá-lo
6. **Callee** retorna controle ao **caller**

Retorno da Subrotina

Ling. alto nível

```
int media(int w, int x, int y, int z) {  
    int result;  
    result = (w + x + y + z) / 4;  
    return result;  
}  
/* Programa principal */  
int main() {  
    int m;  
    m = media(2,3,6,2);  
    ....  
}
```

A blue line starts from the 'return result;' statement in the 'media' function, extends horizontally to the right, then turns 90 degrees downward, and finally turns 90 degrees left to point at the 'm = media(2,3,6,2);' line in the 'main' function, illustrating the return of control.

Controle deve
voltar à instrução
após chamada

Instrução Para Retorno de Subrotinas

- MIPS oferece uma instrução que pode ser utilizado para retornar da subrotina

Jump Register

- Instrução para retornar da subrotina possui um operando:
Registrador que contém um endereço

```
jr registrador
```

- Instrução pula para endereço armazenado no registrador
No caso de retorno de subrotina, o registrador deve ser o \$ra

```
jr $ra
```

Formato de Instruções

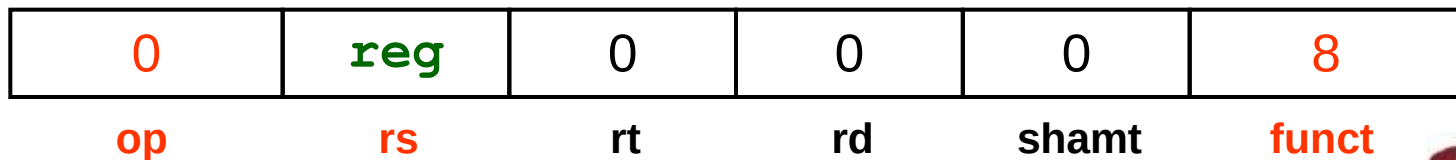
```
jal label
```

Formato J de Instrução



```
jr reg
```

Formato R de Instrução



Código Assembly

```
→media:add $t0,$a0,$a1 # $t0 = w + x  
    add $t1,$a2,$a3 # $t1 = y + z  
    add $v0,$t0,$t1 # $v0 = w + x + y + z  
    srl $v0,$v0,2    # $s0 = (w + x + y + z)/4  
    jr $ra #retornando para caller  
# Programa principal  
main: addi $a0, $zero,2 #$a0 o 1° argumento  
    addi $a1, $zero,3 #$a1 o 2° argumento  
    addi $a2, $zero,6 #$a2 o 3° argumento  
    addi $a3, $zero,2 #$a3 o 4° argumento  
→jal media #chamando media  
→add $s0,$zero,$v0 # $s0 = media(2,3,6,2)
```


E Se Precisarmos de Mais Registradores?

- É comum, precisar-se em uma subrotina de mais registradores que os específicos para as subrotinas

No MIPS:

- 4 para argumentos e 2 para valores retornados

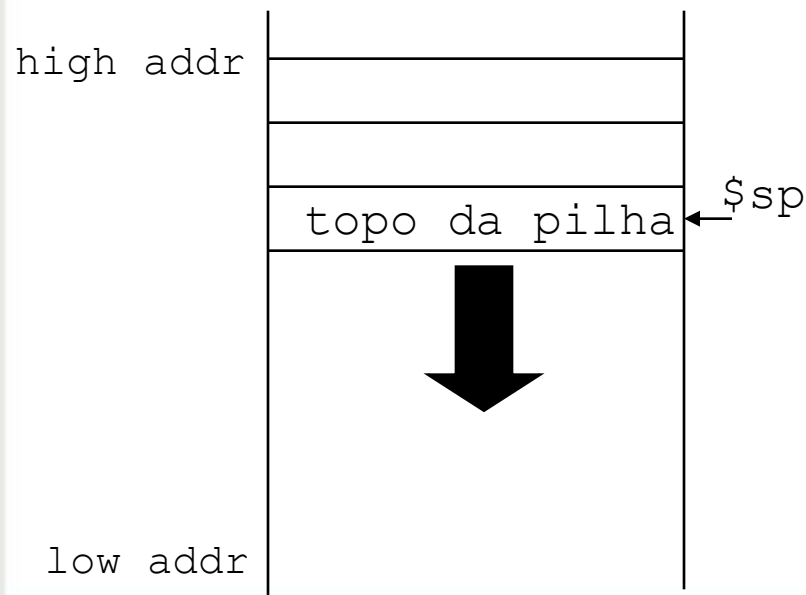
- Solução: **Pilha!**
- No MIPS, registradores que estão em uso fora da subrotina são salvos em uma pilha na memória
- Subrotina utiliza registradores cujos conteúdos foram salvos na pilha

Terminada a subrotina os valores antigos dos registradores são restaurados

Implementando a Pilha

- Utiliza-se parte da memória como pilha
- Pilha cresce do maior para o menor endereço
- Um registrador guarda o endereço do topo

No MIPS: $\$sp$ - *stack pointer* (número 29)



Push

- $\$sp = \$sp - 4$
- Dado inserido no novo $\$sp$

Pop

- Dado removido de $\$sp$
- $\$sp = \$sp + 4$

Usando a Pilha – Salvando Registradores

Código C

```
int media(int w, int x, int y, int z) {  
    int result;  
    result = (w + x + y + z)/4;  
    return result;  
}
```



Código Assembly MIPS

```
addi $sp,$sp,-12 # reservando lugar para 3 itens  
sw $t1,8($sp) # salvando $t1 para uso posterior  
sw $t0,4($sp) # salvando $t0 para uso posterior  
sw $s0,0($sp) # salvando $s0 para uso posterior
```

Usando a Pilha – Executando Subrotina

Código C

```
int media(int w, int x, int y, int z) {  
    int result;  
    result = (w + x + y + z) / 4;  
    return result;  
}
```



Código Assembly MIPS

```
add $t0,$a0,$a1 # $t0 = w + x  
add $t1,$a2,$a3 # $t1 = y + z  
add $s0,$t0,$t1 # $s0 = w + x + y + z  
srl $s0,$s0,2    # $s0 = (w + x + y + z) / 4  
add $v0,$s0,$zero # $v0 = $s0
```

Usando a Pilha – Restaurando Registradores e Retornando

Código C

```
int media(int w, int x, int y, int z) {  
    int result;  
    result = (w + x + y + z) / 4;  
    return result;  
}
```



Código Assembly MIPS

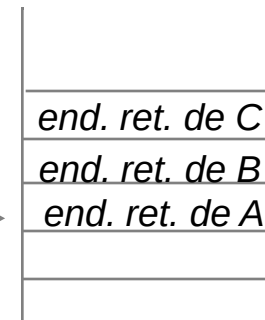
```
lw $s0, 0($sp) # restaurando $s0  
lw $t0, 4($sp) # restaurando $t0  
lw $t1, 8($sp) # restaurando $t1  
addi $sp, $sp, 12 # ajustando topo da pilha  
jr $ra           # retornando a quem chamou
```

Subrotinas Aninhadas

- Como executar subrotinas aninhadas?
 - Funções que chamam outras
 - Funções recursivas
- Subrotina (Caller) que chama outra armazena na pilha:
 - seu endereço de retorno
 - registradores que utilize após o término da subrotina chamada

```
void C () {  
    B();  
}  
void B() {  
    A();  
}
```

*topo da
pilha*



Exemplo de Subrotina Aninhada

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

- Parâmetro **n** em **\$a0**
- Resultado em **\$v0**

Assembly de Subrotina Aninhada

fact:

```
    addi $sp, $sp, -8      # ajustando pilha p/ 2 itens
    sw    $ra, 4($sp)      # salva endereço de retorno
    sw    $a0, 0($sp)      # salva argumento
    slti  $t0, $a0, 1      # testa n < 1
    beq   $t0, $zero, L1   # se sim, resultado é 1
    addi  $v0, $zero, 1     #   pop 2 itens da pilha
    addi  $sp, $sp, 8       #   e retorna
    jr    $ra
L1: addi  $a0, $a0, -1      # senão decrementa n
    jal   fact              #   chamada recursiva
    lw    $a0, 0($sp)      # restaura n original
    lw    $ra, 4($sp)      #   e endereço de retorno
    addi  $sp, $sp, 8       # pop 2 itens da pilha
    mul   $v0, $a0, $v0     # multiplica p/ obter resultado
    jr    $ra              # e retorna
```


Organização da Memória

- Segmento de texto (text) : código
- Dados estáticos (static data): variáveis globais

Variáveis estáticas

No MIPS, registrador **\$gp** guarda o início do segmento

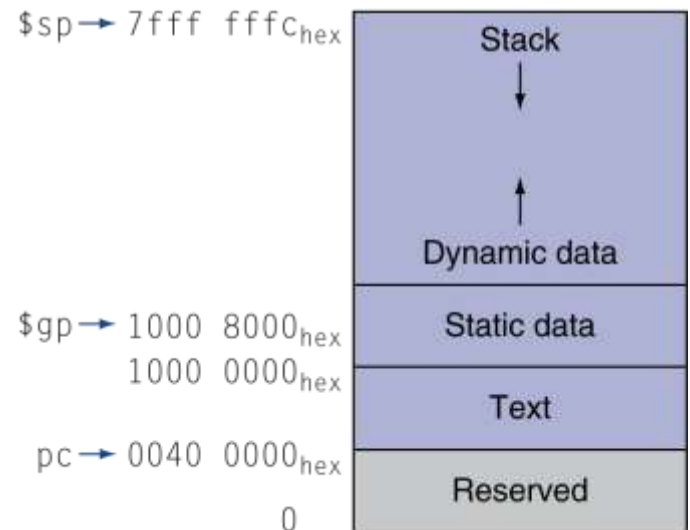
- Dados dinâmicos : heap

Exs: malloc em C, new em Java

- Pilha (Stack)

Variáveis locais

Registradores



Chamada de Subrotina em Outras Arquiteturas

- Instruções:

call

- empilha endereço de retorno
- muda fluxo de controle

ret

- recupera endereço de retorno

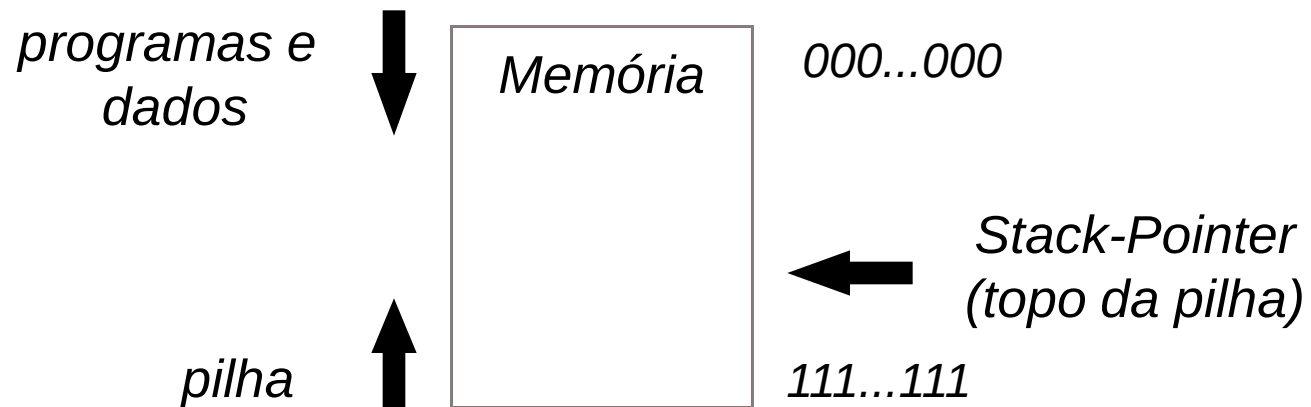
- Outras instruções de suporte...

Salvar todos registradores na pilha

alocar parte da pilha para armazenar variáveis locais e parâmetros

Usando a Pilha em Outras Arquiteturas

- Utiliza parte da memória como pilha



SP: Registrador adicional

- Instruções adicionais:

push reg: mem(SP) reg ; decrementa SP

pop reg: incrementa SP, reg mem(SP);

MIPS vs. Outras arquiteturas

- Endereço de retorno:
 - MIPS: registrador
 - Outras: Memória
 - Melhor desempenho
- Acesso à Pilha:
 - MIPS: instruções lw e sw
 - Outras: instruções adicionais
 - Menor complexidade na implementação
 - Compilador mais complexo
- Chamadas aninhadas ou recursivas
 - MIPS: implementada pelo compilador
 - Outras: suporte direto da máquina
 - Compilador mais complexo

Modos de Endereçamento do MIPS

- Modo de endereçamento se refere às maneiras em que instruções de uma arquitetura especificam a localização do operando
 - Onde e como pode ser acessado
- No MIPS, operandos podem estar em:
 - Registradores
 - Memória
 - Na própria instrução

Endereçamento de Registrador

■ Operações aritméticas:

O operando está em um registrador e a instrução contem o número do registrador

add R3 ,R3 ,R7



R3 <- R3 + R7

Instrução

00101010	00011	00111	00011
Opcode	Oper. 1	Oper.2	Destino

	R0
	R1
	R2
00...0001 / 00...00110	R3
	R4
	R5
	R6
00...000101	R7

Registradores

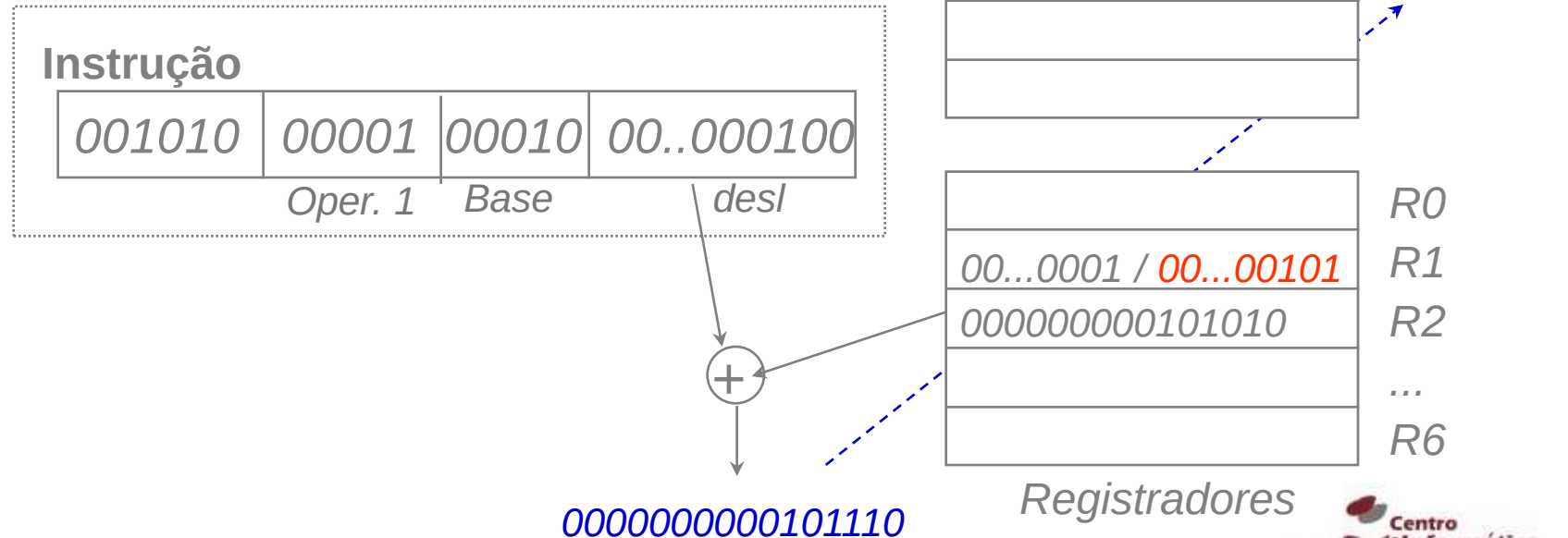
Endereçamento Base

■ Instruções de acesso à memória:

Instrução: deslocamento

Registrador de base: end- inicial

lw R1, des1(R2)



Endereçamento imediato

- Operações aritméticas e de comparação:

O operando é especificado na instrução

addi R1,R2, 12

Instrução

001011	00010	00001	000...01100
--------	-------	-------	-------------

Oper.1

Destino

	R0
00...0000 / 00...01100	R1
00...0000	R2
	R3
	R4
	R5
	R6
	R7

Registradores

Endereçamento (Pseudo)Direto

■ Instrução de Desvio Incondicional:

o (pseudo)endereço da próxima instrução (endereço da palavra) é especificado na instrução

4 bits mais significativos do PC são concatenados ao endereço especificado multiplicado por 4

j endereço



PC ← conc (PC (4 bits) , endereço* 4)

Instrução

001010	000000000000000000000000101010
--------	--------------------------------

PC=0001..101110

conc

*4

0001..10101000

Add R1, R1, R3
J 000000.....101010

PC=00..10101000

PC=0001..101110

Endereçamento Relativo a PC

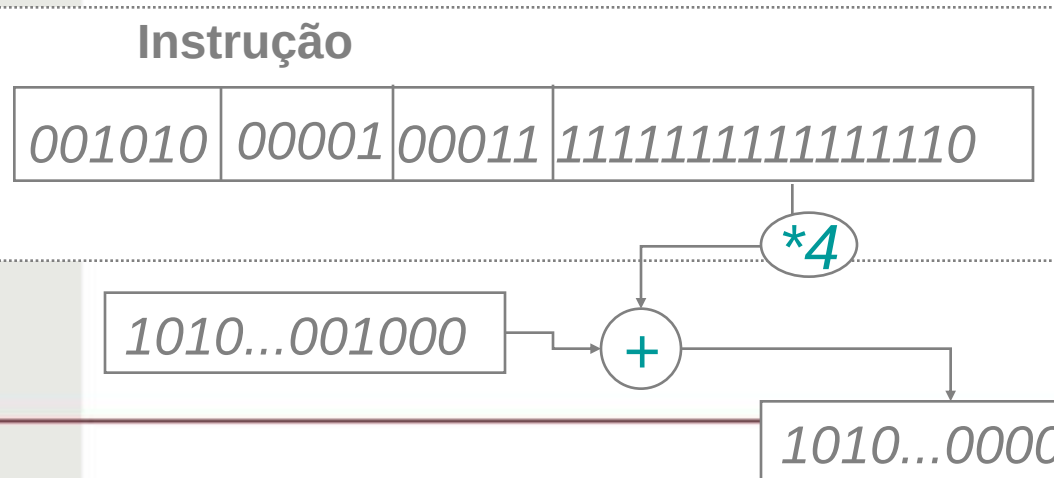
■ Instrução de Desvio Condicional:

o número de instruções a serem puladas a partir da instrução é especificado na instrução

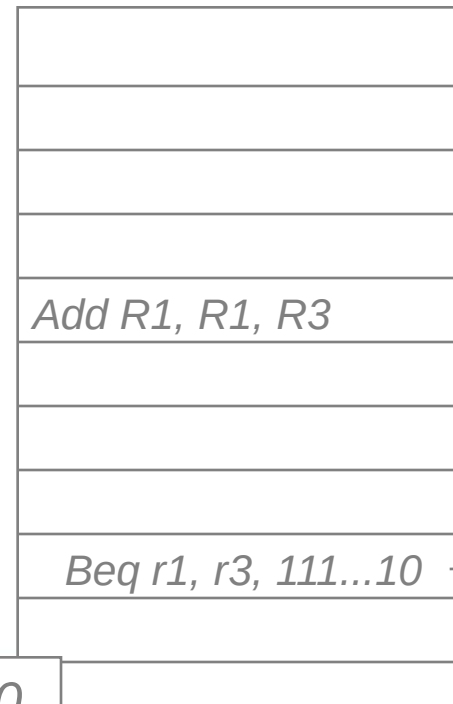
beq R1 ,R2 ,des1



PC <- PC + des1 * 4



Memória



PC=1010...0000

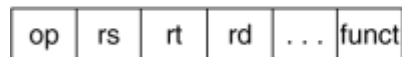
PC=1010...1000

Resumo dos Modos de Endereçamento do MIPS

1. Immediate addressing



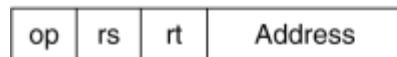
2. Register addressing



Registers

Register

3. Base addressing



Register

+

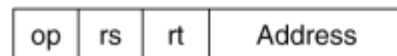
Memory

Byte

Halfword

Word

4. PC-relative addressing



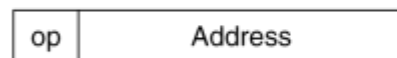
PC

+

Memory

Word

5. Pseudodirect addressing



PC

:

Memory

Word

Tipos de Datos

■ Escalar

números

- Inteiros
- Ponto-Flutuante (real)

caracteres

- ASCII
- EBCDIC

dados lógicos

Inteiros

■ Representação binária

- sinal-magnitude

00000000000001010

+ 10

10000000000001010

- 10

- complemento a 1

00000000000001010

+ 10

1111111111110101

- 10

- complemento a 2

00000000000001010

+ 10

111111111110110

- 10

Dados lógicos

■ Representação

Uma palavra

Um bit

0000000000000000

Falso

0000000000000001

Verdadeiro

0000000000010000

Verdadeiro

Falso

Ponto Flutuante

■ Representação

$$3,14 = 0,314 \times 10^1 = 3,14 \times 10^0$$

$$0,000001 = 0,10 \times 10^{-5} = 1,00 \times 10^{-6}$$

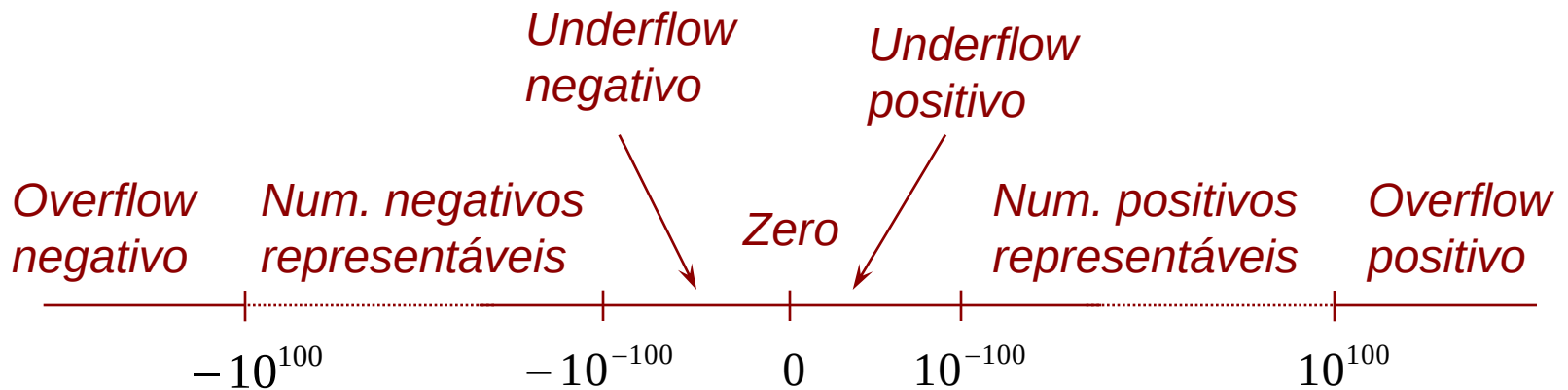


$$n = f \times 10^{e-}$$

<i>sinal</i>	<i>expoente</i>	<i>mantissa (fração)</i>
--------------	-----------------	--------------------------

Ponto Flutuante

■ Padrão IEEE



Item	Precisão simples	Precisão dupla
Sinal	1	1
Expoente	8	11
Mantissa	23	52
Total	32	64

Usando o Simulador MIPSIT

■ Simulando um programa

O que é visível:

- Registradores (CPU)
 - Conteúdo em hexadecimal
 - Associação nome e número
- Memória (RAM)
 - Quatro colunas:
 - Endereço
 - Conteúdo (hexa)
 - Rótulos
 - Instrução de máquina

Resumo de Instruções do MIPS

Instrução	Descrição
nop	No operation
lw reg, end(reg_base)	reg. = mem (reg_base+end)
sw reg, end(reg_base)	Mem(reg_base+end) = reg
add regi, regj, regk	Regi. <- Regj. + Regk
sub regi, regj, regk	Regi. <- Regj. - Regk
and regi, regj, regk	Regi. <- Regj. and Regk
xor regi, regj, regk	Regi = regj xor regk
srl regd, regs, n	Desloca regs para direita n vezes sem preservar sinal, armazena valor deslocado em regd.
sra regd, regs, n	Desloca regs para dir. n vezes preservando o sinal, armazena valor deslocado em regd.
sll regd, regs, n	Desloca regs para esquerda n vezes, armazena valor deslocado em regd.
ror regd, regs, n	Rotaciona regs para direita n vezes, armazena valor deslocado em regd.
rol regd, regs, n	Rotaciona regs para esquerda n vezes, armazena valor deslocado em regd.
beq regi, regj, desl	PC = PC + desl*4 se regi = regj
bne regi, regj, desl	PC = PC + desl *4 se regi <> regj
slt regi, regj, regk	Regi =1 se regj < regk senão regi=0
j end	Desvio para end
jr reg	Pc = reg
jal end	Reg31 = pc, pc = endereço
break	Para a execução do programa