



## Estrutura do tema ISA do IA-32

1. Desenvolvimento de programas no IA-32 em Linux
2. Acesso a operandos e operações
3. Suporte a estruturas de controlo
4. Suporte à invocação/regresso de funções
5. Análise comparativa: IA-32 (CISC) e MIPS (RISC)
6. Acesso e manipulação de dados estruturados



## RISC versus IA-32 :

- RISC: conjunto reduzido e simples de instruções
  - pouco mais que o *subset* do IA-32 já apresentado...
  - instruções simples, mas eficientes
- operações aritméticas e lógicas:
  - 3-operandos (RISC) versus 2-operandos (IA-32)
  - RISC: operandos sempre em registos, 32 registos genéricos visíveis ao programador, sendo normalmente
    - 1 reg apenas de leitura, com o valor 0
    - 1 reg usado para guardar o endereço de regresso da função
    - 1 reg usado como *stack pointer* (*s/w*)

— . . . .



## RISC versus IA-32 (cont.):

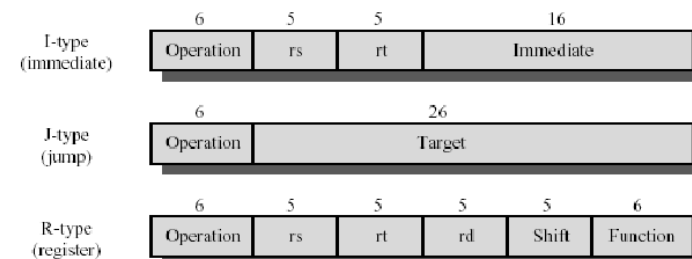
- RISC: modos simples de endereçam. à memória
  - apenas 1 modo de especificar o endereço:  
 $\text{Mem}[C^{te} + (\text{Reg}_b)]$  ou  $\text{Mem}[(\text{Reg}_b) + (\text{Reg}_i)]$
  - 2 ou 3 modos de especificar o endereço:  
 $\text{Mem}[C^{te} + (\text{Reg}_b)]$  e/ou  
 $\text{Mem}[(\text{Reg}_b) + (\text{Reg}_i)]$  e/ou  
 $\text{Mem}[C^{te} + (\text{Reg}_b) + (\text{Reg}_i)]$
- RISC: uma operação elementar por ciclo máquina
  - por ex. *push/pop* (IA-32) substituído por par de instruções *sub&store/load&add* (RISC)

— . . . .



## RISC versus IA-32 (cont.):

- RISC: formatos simples de instruções
  - comprimento fixo e poucas variações
  - ex.:MIPS



Funções em assembly:  
IA-32 versus MIPS (RISC) (1)

Principal diferença:

- na organização dos registos
  - IA-32: poucos registos genéricos => variáveis e argumentos normalmente na *stack*
  - RISC: 32 registos genéricos => registos para variáveis locais, & registos para passagem de argumentos & registo para endereço de regresso
- consequências:
  - menor utilização da *stack* nas arquiteturas RISC
  - RISC potencialmente mais eficiente

Análise de um exemplo (swap) ...

Funções em assembly:  
IA-32 versus MIPS (RISC) (2)

IA-32	MIPS
<pre> _swap:   pushl   %ebp   movl    %esp, %ebp   pushl   %ebx   movl    8(%ebp), %edx   movl    12(%ebp), %ecx   movl    (%edx), %ebx   movl    (%ecx), %eax   movl    %eax, (%edx)   movl    %ebx, (%ecx)   popl    %ebx   popl    %ebp   ret  _call_swap:   pushl   %ebp   movl    %esp, %ebp   subl    \$24, %esp   movl    \$15213, -4(%ebp)   movl    \$91125, -8(%ebp)   leal    -4(%ebp), %eax   movl    %eax, (%esp)   leal    -8(%ebp), %eax   movl    %eax, 4(%esp)   call    _swap   movl    %ebp, %esp   popl    %ebp   ret                     </pre>	<pre> swap:   lw      \$v1, 0(\$a0)   lw      \$v0, 0(\$a1)   sw      \$v0, 0(\$a0)   sw      \$v1, 0(\$a1)   j       \$31  call_swap:   subu    \$sp, \$sp, 32   sw      \$ra, 24(\$sp)   li      \$v0, 15213   sw      \$v0, 16(\$sp)   li      \$v0, 0x10000   ori     \$v0, \$v0, 0x63f5   sw      \$v0, 20(\$sp)   addu    \$a0, \$sp, 16 # &amp;zip1= sp+16   addu    \$a1, \$sp, 20 # &amp;zip2= sp+20   jal     swap   lw      \$ra, 24(\$sp)   addu    \$sp, \$sp, 32   j       \$ra                     </pre>

Funções em assembly:  
IA-32 versus MIPS (RISC) (3)

**call\_swap**

1. Invocar *swap*

- salvar registos
- passagem de argumentos
- chamar rotina e guardar endereço de regresso

**IA-32**

```

leal -4(%ebp), %eax
pushl %eax
leal -8(%ebp), %eax
pushl %eax
call swap
                    
```

*Não há reg para salvag.*  
Calcula &zip2  
Push &zip2  
Calcula &zip1  
Push &zip1  
Invoca swap

**Acessos à stack**

**MIPS**

```

sw      $ra, 24($sp)
addu   $a0, $sp, 16
addu   $a1, $sp, 20
jal     swap
                    
```

Salvag. reg c/ ender. regresso  
Carrega no reg &zip1  
Carrega no reg &zip2  
Invoca swap

Funções em assembly:  
IA-32 versus MIPS (RISC) (4)

**swap**

1. Inicializar *swap*

- atualizar *frame pointer*
- salvar registos
- reservar espaço p/ locais

**IA-32**

```

swap:
  pushl %ebp
  movl  %esp, %ebp
  pushl %ebx
                    
```

Salvag. antigo %ebp  
%ebp novo frame pointer  
Salvag. %ebx  
Não é preciso esp. p/ loc.

**Acessos à stack**

**MIPS**

Frame pointer p/ atualiz: NÃO  
Registos p/ salvag: NÃO  
Espaço p/ locais: NÃO

Funções em assembly:  
IA-32 versus MIPS (RISC) (5)

Funções em assembly:  
IA-32 versus MIPS (RISC) (6)

2. Corpo de swap ...

```
movl 12(%ebp), %ecx  Get yp
movl 8(%ebp), %edx   Get xp
movl (%ecx), %eax    Get y
movl (%edx), %ebx    Get x
movl %eax, (%edx)    Armazena y em *xp
movl %ebx, (%ecx)    Armazena x em *yp
```

IA-32

Acessos à memória (todas...)

MIPS

```
lw $v1, 0($a0)  Get x
lw $v0, 0($a1)  Get y
sw $v0, 0($a0)  Armazena y em *xp
sw $v1, 0($a1)  Armazena x em *yp
```

3. Término de swap ...

```
popl %ebx  Recupera %ebx
movl %ebp, %esp  Recupera %esp
popl %ebp  Recupera %ebp
ret        Volta à função chamadora
```

IA-32

Acessos à stack

MIPS

```
j $31
```

Espaço a libertar de var locais: NÃO  
Recuperação de registos: NÃO  
Recuperação do frame ptr: NÃO  
Volta à função chamadora

Funções em assembly:  
IA-32 versus MIPS (RISC) (7)

2. Terminar invocação de swap ...

```
addl $8, (%esp)  Actualiza stack pointer
                Não há reg's a recuperar
```

IA-32

Acessos à stack

MIPS

```
lw $ra, 24($sp)  Recupera reg c/ ender regresso
```

Espaço a libertar na stack: NÃO