

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

Análise do nível ISA: o modelo RISC versus IA-32 (1)

RISC versus IA-32 :

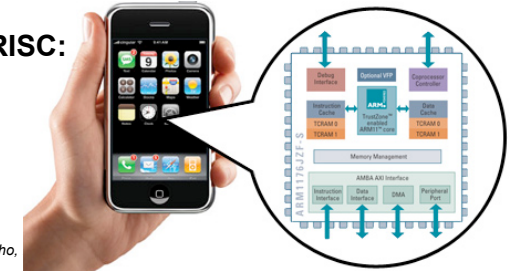
- RISC: conjunto reduzido e simples de instruções
 - pouco mais que o *subset* do IA-32 já apresentado...
 - instruções simples, mas eficientes
- operações aritméticas e lógicas:
 - 3-operandos (RISC) versus 2-operandos (IA-32)
 - RISC: operandos sempre em registos,
 - 32 registos genéricos visíveis 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)

—

Caracterização das arquiteturas RISC

- conjunto reduzido e simples de instruções
- formatos simples de instruções
- operandos sempre em registos
- modos simples de endereçamento à memória
- uma operação elementar por ciclo máquina

Exemplo de um chip RISC: ARM



Análise do nível ISA: o modelo RISC versus IA-32 (2)

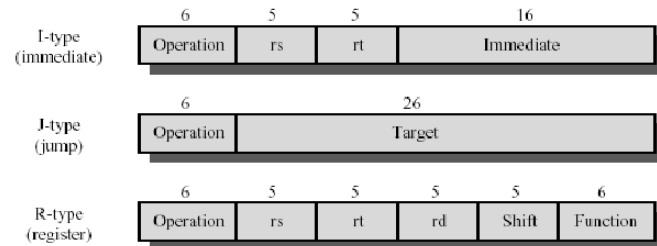
RISC versus IA-32 (cont.):

- RISC: modos simples de endereçamento à memória
 - apenas 1 modo de especificar o endereço:
 $Mem[C^{te} + (Reg_b)]$ ou $Mem[(Reg_b) + (Reg_i)]$
 - 2 ou 3 modos de especificar o endereço:
 $Mem[C^{te} + (Reg_b)]$ e/ou
 $Mem[(Reg_b) + (Reg_i)]$ e/ou
 $Mem[C^{te} + (Reg_b) + (Reg_i)]$
- RISC: uma operação elementar por ciclo máquina
 - por ex. *push/pop* (IA-32) substituído pelo 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



Principal diferença na implementação de funções:

- 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 arquitecturas RISC
 - RISC potencialmente mais eficiente

Análise de um exemplo (swap) ...

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

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

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 # &zip1= sp+16 addu \$a1, \$sp, 20 # &zip2= sp+20 jal swap lw \$ra, 24(\$sp) addu \$sp, \$sp, 32 j \$ra </pre>

call_swap

1. Invocar swap

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

IA-32

```

leal   -4(%ebp), %eax   Calcula &zip2
pushl  %eax             Push &zip2
leal   -8(%ebp), %eax   Calcula &zip1
pushl  %eax             Push &zip1
call   swap             Invoca swap
                
```

MIPS

```

sw     $ra, 24($sp)     Salvag. reg c/ ender. regresso
addu   $a0, $sp, 16    Carrega no reg &zip1
addu   $a1, $sp, 20    Carrega no reg &zip2
jal    swap            Invoca swap
                
```

Acessos à stack

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

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

swap

1. Inicializar swap

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

```

swap:
  pushl %ebp          Salvag. antigo %ebp
  movl  %esp, %ebp   %ebp novo frame pointer
  pushl %ebx          Salvag. %ebx
  
```

IA-32

Acessos à stack

MIPS

Frame pointer p/ atualizar: NÃO
Registos p/ salvar: NÃO
Espaço p/ locais: NÃO

Não é preciso espaço p/ locais

swap

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
  
```

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

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

swap

3. Término de swap ...

- libertar espaço de var locais
- recuperar registos
- recuperar antigo *frame pointer*
- voltar a *call_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

Espaço a libertar de var locais: NÃO
Recuperação de registos: NÃO
Recuperação do *frame ptr*: NÃO

Não há espaço a libertar

```

j $31  Volta à função chamadora
  
```

call_swap

2. Terminar invocação de swap ...

- libertar espaço de argumentos na *stack*...
- recuperar registos

```

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

IA-32

Acessos à stack

MIPS

Espaço a libertar na *stack*: NÃO

```

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