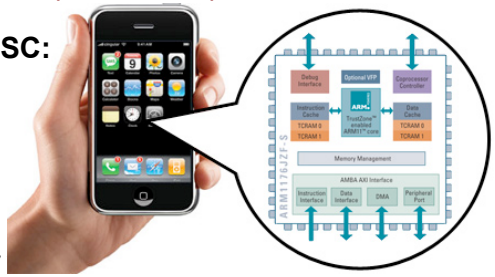
Estrutura do tema ISA do IA-32

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

Caracterização das arquitecturas 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



RISC *versus* IA-32 :

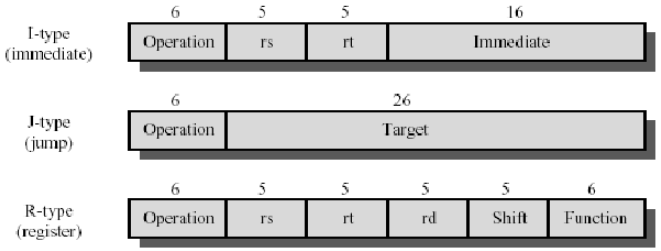
- RISC: conjunto reduzido e simples de instruções
  - pouco mais que o *subset* do IA-32 já apresentado...
  - instruções simples, mas eficientes
- operações aritméticas e lógicas:
  - 3-operandos (RISC) *versus* 2-operandos (IA-32)
  - RISC: operandos sempre em registos,
    32 registos genéricos visíveis 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 endereçamento à memória
  - apenas 1 modo de especificar o endereço:
    `Mem[Cte+(Regb)]` ou `Mem[(Regb)+(Regi)]`
  - 2 ou 3 modos de especificar o endereço:
    `Mem[Cte+(Regb)]` e/ou
    `Mem[(Regb)+(Regi)]` e/ou
    `Mem[Cte+(Regb)+(Regi)]`
- 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 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`) ...

IA-32

```
_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
```

MIPS

```
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
```


`call_swap`

1. Invocar `swap`
   - salvaguardar registos
   - passagem de argumentos
   - chamar rotina e guardar endereço de regresso

IA-32

```
        Não há reg para salvag.
leal    -4(%ebp),%eax   Calcula &zip2
pushl   %eax            Push &zip2
leal    -8(%ebp),%eax   Calcula &zip1
pushl   %eax            Push &zip1
call    swap            Invoca swap
```

Acessos à *stack*

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
```

swap

1. Inicializar `swap`
- actualizar *frame pointer*
- salvaguardar registos
- reservar espaço p/ locais

IA-32

```
swap:
  pushl  %ebp          Salvag. antigo %ebp
  movl   %esp,%ebp     %ebp novo frame pointer
  pushl  %ebx          Salvag. %ebx
                 Não é preciso esp. p/ loc.
```

Acessos à *stack*

MIPS

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

swap

2. Corpo de `swap` ...

IA-32

```
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
```

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
```


swap

3. Término de `swap` ...
- libertar espaço de var locais
- recuperar registos
- recuperar antigo *frame pointer*
- voltar a `call_swap`

IA-32

```
                Não há espaço a libertar
popl  %ebx          Recupera %ebx
movl  %ebp,%esp     Recupera %esp
popl  %ebp          Recupera %ebp
ret                 Volta à função chamadora
```

Acessos à *stack*

MIPS

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

```
j   $31     Volta à função chamadora
```


`call_swap`

2. Terminar invocação de `swap` . . .
- libertar espaço de argumentos na *stack*...
- recuperar registos

IA-32

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

Acessos à *stack*

MIPS

Espaço a libertar na stack: NÃO

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