Arquitetura de Computadores

MIEI – 2017/18
DI-FCT/UNL
Aula 11

Saltos condicionais (inteiros c/sinal)

- Saltos condicionais para as relações entre inteiros com sinal:

após *sub X, Y* ou *cmp X,Y*

salta se			*flags*
Y>X	greater than	JG/JNLE	SF = OF and ZF=0
Y>=X	greater or equal	JGE/JNL	SF = OF
Y<X	less than	JL/JNGE	SF <> OF and ZF=0
Y<=X	less or equal	JLE/JNG	ZF=1 or SF <> OF

Saltos

- Instruções de salto de acordo com os valores de "flags"

inst.	Condition *(C expr.)*	Description
`jmp`		Unconditional
`je/jz`	`ZF`	Equal / Zero
`jne/jnz`	`~ZF`	Not Equal / Not Zero
`js`	`SF`	Negative
`jns`	`~SF`	Nonnegative
`jg`	`~(SF^OF)&~ZF`	Greater (Signed)
`jge`	`~(SF^OF)`	Greater or Equal (Signed)
`jl`	`(SF^OF)`	Less (Signed)
`jle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`ja`	`~CF&~ZF`	Above (unsigned)
`jb`	`CF`	Below (unsigned)

- Exemplos de construções de C

Exemplo: if

- como executar código diferente dependendo de uma condição?
  - condição → estado das flags
  - salto condicional (jxx)

if (*condicao*)
  codigo_do_then;
else
  codigo_do_else;

```
# código que altere as flags
# (exemplo cmp $5, %eax)
jxx bloco_else
        #  codigo_do_then
        jmp fim_if
bloco_else:
        # codigo_do_else
fim_if:
```


Exemplo: for

- ciclo com base num contador

for (i=*0*; i<n; i++)
  codigo_do_ciclo;

```
#exemplo:
        mov $0, %ecx
inicio_for:
        cmp $n, %ecx
        je fim_for
        #  codigo_do_ciclo
        inc %ecx
        jmp inicio_for
fim_for:
```

Exemplo: while

- ciclo com teste "à cabeça"

while (*condicao*)
 codigo_do_ciclo;

```
inicio_while:
# código que altere as flags
# (exemplo cmp $5, %eax)
      jxx fim_while
      #  codigo_do_ciclo
      jmp inicio_while
fim_while:
```


Exemplo: do-while

- ciclo com teste "no fim"

do
 codigo_do_ciclo;
while (*condicao*);

```
inicio_do:
  # codigo_do_ciclo

  # código que altere as flags
  # (exemplo cmp $5, %eax)
  jxx inicio_do
```


Exemplo: incr. elementos de um vetor

```
.data
vet: .int 0, 3, 5, 10, 2

.text
      . . .
      mov $5, %eax
      mov $vet, %ebx
ciclo: incl (%ebx)#incrementar todos os ints de vet
       add $4, %ebx
       dec %eax
       jnz ciclo
       . . .
```


Exemplo: for (2)

- ciclo com base num contador

for (i=*n*; i>0; i--)
 codigo_do_ciclo;

```
#exemplo:
          mov $n, %ecx
          cmp $0, %ecx
inicio_for:
          jz fim_for
          #  codigo_do_ciclo
          dec %ecx
          jmp inicio_for
fim_for:
```


LOOP

- Salto com base num contador, para ciclos que executam pelo menos uma vez
  - equivale a 'dec %cx' seguido de 'jnz'
- Usa **implicitamente** o registo CX *(16bits)*

for (i=*n*; i>0; i--)
 codigo_do_ciclo;

```
          mov $n, %cx
inicio_for:
          #  codigo_do_ciclo
          loop inicio_for
```

Nota: para n > 0!

Instruções lógicas

- Tratam os *bytes* como vetores de bits
  - Onde cada bit: 1=true, 0=false
  - as *flags* refletem o resultado
- Exemplos de instruções:

not *X*	**and** *X,Y*	**or** *X,Y*	**xor** *X,Y*
X←not X	*Y←X and Y*	*Y←X or Y*	*Y←X xor Y*

Em C ou Java:

X=~X	*Y=Y&X*	*Y=Y|X*	*Y=X^Y*

- test X, Y
  - equivale ao AND sem alterar o 2º operando
  - Exemplo:
    test 0b01010101, %ah
  - põe ZF a 1 se os bits 0,2,4 e 6 de AH estão a 0, mas não altera AH
  - Ou seja, testa se algum dos bits indicados está a 1
  - ou se todos estão a zero

Manipular bits

- Exemplos:
  - not %al -- inverte cada bit de AL
  - and 0x7FFF, %ax -- põe a 0 o bit 15 de ax
  - or 0x8000, %ax -- põe a 1 o bit 15 de ax
  - xor 0x8000, %ax -- inverte o bit 15 de ax
  - test 0b00000100, %dl -- testa bit 2 de DL: põe ZF a 1 se o bit 2 de DL for 0; ZF fica 0 se o bit 2 de DL for 1

Deslocamentos (shifts)

- shl n, X (C ou Java: X=X<<n)
  - desloca os bits em X para a esquerda (se inteiro é como multiplicar por 2)
    CF <----| <----X------ |<---- 0
  - Exemplo:

mov 0b00110001, %al
shl $1, %al
ou **shl %al** AL fica com 0b01100010

Deslocamentos (shifts)

- shr n, X (C se X unsigned int: X=X>>n. Java: X=X>>>n.)
  - Desloca os bits em X para a direita (se inteiro é como dividir por 2)
    0 ---->| -----X-----> |----> CF
  - Exemplo:

mov 0b00110001, %al
shr $1, %al
ou **shr %al** AL fica com 0b00011000

Deslocamentos (shifts)

- sar n, X (se X é int, C e Java: X=X>>n)
  - Desloca os bits em X para a direita mas mantém o bit mais significativo (se inteiro, é como dividir por 2 mantendo sinal)

Exemplo:
mov 0b**1**0110001, %al
sar $1, %al
ou **sar %al** AL fica com 0b**11**011000

Modos de indicar um operando

- Diferentes modos de indicar os operandos:
  - **imediato** – o operando está na própria instrução (após o fetch já se encontra no CPU)
  - **registo** – o valor encontra-se num registo do CPU; na instrução encontra-se o código (endereço) do registo
  - **endereçamento de memória** – o operando está em memória …
    Mas há muitos modos de endereçar operandos em memória …

Modos de endereçar a memória (1)

- Muitos modos diferentes de indicar o endereço do operando
  - Endereçamento **direto**:
    o endereço efetivo está na instrução. Exemplos:
    mov (100), %eax
    mov var1, %eax
  - Endereçamento **indireto**:
    é obtido pelo CPU na execução da instrução com base em várias informações codificadas na instrução (pode envolver valores na instrução e o conteúdo de registos)

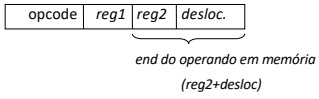
Modos de endereçar a memória (2)

- **Indireto por registo** *(visto antes)*
  - um registo contém o endereço na memória onde está o operando. Exemplo: `mov (%ebx), %eax`
- Outras variantes: envolvendo mais de um registo, deslocamentos e factores de escala
  - . . .
- *Endereçamento **indireto por memória:***
  - *endereço efetivo encontra-se na memória, cujo endereço é o indicado na instrução*

  Não suportado na arquitetura IA-32!

Endereçamento baseado ou indexado

- Indireto com **base e deslocamento** (endereçamento baseado ou indexado)
  - um registo contém um endereço (base) e na instrução é indicado um deslocamento
  - Ou, a base é dada na instrução e o registo contém o deslocamento (índice)
  - exemplo: mov 8(%ebx), %eax

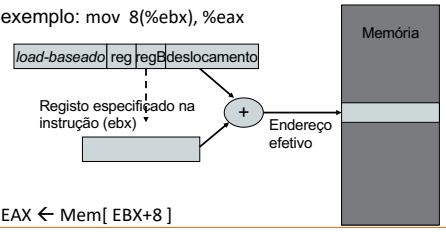


end do operando em memória
(reg2+desloc)

Endereçamento baseado ou indexado (2)

- O endereço efetivo é obtido somando o conteúdo de um registo (registo base ou índice) a uma constante

exemplo: mov 8(%ebx), %eax



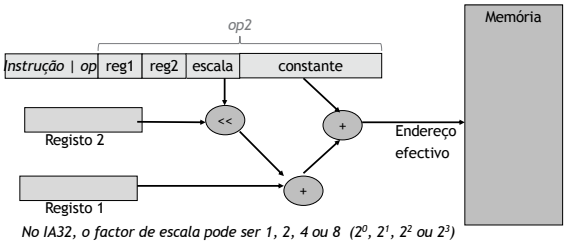
EAX ← Mem[EBX+8]

Endereçamento indireto de memória

- Forma geral em *assembly* para IA-32:
  - **D(Rb, Ri, S)** End. Efetivo = D + Rb + Ri*S
    - D: Constante (deslocamento)
    - Rb: Registo Base: Qualquer dos 8 registos gerais
    - Ri: Registo índice: Qualquer, excepto esp
    - S: Escala: 1, 2, 4 ou 8
- Exemplo: `movl $5, 100(%ebx, %ecx, 2)`

Endereçamento indireto de memória (2)

- O endereço efetivo é obtido somando o conteúdo do registo1 (registo base) ao resultado de aplicar ao registo 2 (registo índice) um factor de escala, e somando a constante



No IA32, o factor de escala pode ser 1, 2, 4 ou 8 (2^0, 2^1, 2^2 ou 2^3)

Endereçamento de memória

- Alguns casos particulares (c/exemplos):

D ou (D)	**`var`** ou **`(var)`**	- direto
(Rb)	**`(%eax)`**	- indireto p/registo
D(Rb)	**`vet(%ebx)`**	- ind baseado/indexado
(Rb, Ri, S)	**`(%ebx, %ecx, 2)`**	- ind baseado e indexado
D(, Ri, S)	**`vet( ,%ecx, 4)`**	

Exemplos de cálculo do endereço efectivo

edx	**0xf000**
ecx	**0x100**

End. assembly	Cálculo	Endereço
`0x8(%edx)`	`0xf000 + 0x8`	`0xf008`
`(%edx, %ecx)`	`0xf000 + 0x100`	`0xf100`
`(%edx,%ecx,4)`	`0xf000 + 4*0x100`	`0xf400`
`0x80( ,ecx,2)`	`0x80 + 2*0x100`	`0x280`