Testes e ciclos

João Canas Ferreira

Fevereiro 2013

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Indicadores de estado (revisão)

Os indicadores de estado *(flags)* são atualizados após a execução de algumas operações (consultar sumário de instruções):

- ZF=1 se resultado da instrução for 0 [zero flag]
- CF=1 se existir transporte [carry flag]
- SF=1 se resultado for negativo [sign flag]
- OF=1 se resultado exceder gama (c/ sinal) [overflow flag]
- PF=1 se número de bits for par (byte LSB) [parity flag]
- AC=1 se existir transporte do bit 3 para o bit 4 [auxiliary carry]

Maneiras de alterar os indicadores de estado

Afetar vários indicadores:

- `CMP op1, op2` é equivalente a `SUB op1, op2`, mas sem afetar op1
- `TEST op1, op2` é equivalente a `AND op1, op2`, mas sem afetar op1

Afetar indicador CF

Este indicador pode ser alterado por instruções específicas:

CLC CF=0 *(clear carry flag)*

STC CF=1 *(set carry flag)*

CMC CF=NOT(CF) *(complement carry flag)*

BT Copia o bit N de um operando para CF *(Bit test)*

- Formato: `BT bitBase, N`
- `bitBase` pode ser reg16/mem16 ou reg32/mem32
- N pode ser reg16, reg32 ou imm8 (constante de 8 bits)

Utilização dos indicadores de estado

- Os indicadores de estado são usados para determinar se um salto condicional é tomado
  - `J<cond> destino`
  - O destino do salto relativo é especificado em bytes, contados a partir da instrução seguinte
  - <cond> representa a condição (E,C, NE, NC, etc.)
- Os indicadores podem ser usados para colocar o byte menos significativo a 0 (00000000b) ou a 1 (00000001b)
  - `SET<cond> reg8/mem8`
  - Exemplo: `SETZ BL` coloca BL=1 se ZF=1
- A instrução `LAHF` copia o LSB de EFLAGS para registo AH
  - O LSB contém: SF, ZF, AC, PF e CF
- A instrução `SAHF` copia registo AH para EFLAGS

Transferências condicionais

- Uma instrução de transferência condicional executa a transferência apenas se a condição for verdadeira.
- Formato: `CMOV<cond> reg, reg/mem`
- São apenas suportados operandos de 16 ou 32 bits.

Exemplo: `varA = varB > varC ? varD : varE`

Scheme: `(set! varA (if (> varB varC) varD varE))`

```
        mov   eax, varB
        cmp   eax, varC          mov     eax, varB
        jng   L1                 cmp     eax, varC
        mov   eax, varD    ⇒     mov     eax, varD
        jmp   L2                 cmovng  eax, varE
L1:     mov   eax, varE          mov     varA, eax
L2:     mov   varA, eax
```


Instruções de ciclo

Ciclos usam frequentemente um contador de iterações.

Existem instruções para implementar simultaneamente o ajuste do contador, o teste e o salto condicional.

- `LOOP destino`
  - ECX = ECX - 1; se ECX $\neq$ 0, saltar para `destino` (relativo)
- Em muitos casos, é útil incluir mais um teste
- `LOOPE destino` ou `LOOPZ destino`
  - ECX = ECX - 1; se ECX $\neq$ 0 e ZF=1, saltar para `destino`
- `LOOPNE destino` ou `LOOPNZ destino`
  - ECX = ECX - 1; se ECX $\neq$ 0 e ZF=0, saltar para `destino`
- O decremento de ECX não afeta os indicadores
- Estas instruções usam obrigatoriamente o registo ECX

Exemplo de utilização de LOOPNZ (com erro!)

O fragmento seguinte pretende determinar o valor do primeiro elemento não-negativo da sequência (fica guardado em EAX).

```
            .data
seq         SWORD  -3,-6,-1,-10,10,30,40,4
            .code
            mov    esi,OFFSET seq     ; endereço inicial
            mov    ecx,LENGTHOF seq   ; nº de elementos de seq
prox:       test   SWORD PTR [esi],8000h
            ; ZF=1 indica que bit de sinal é 0 (positivo)
            add    esi, 2             ; afeta ZF!!
            loopnz prox               ; repete
            jnz    nada               ; não encontrou!
            mov    ax,[esi-2]         ; o valor pretendido
            jmp    tratar_valor
nada:       ...
```


Exemplo de utilização de LOOPNZ (correto)

Versão corrigida do fragmento anterior.

```
            .data
seq         SWORD  -3,-6,-1,-10,10,30,40,4
            .code
            mov    esi,OFFSET seq
            mov    ecx,LENGTHOF seq
prox:       add    esi,2
            test   SWORD PTR [esi-2],8000h
            loopnz prox               ; depende de test
            jnz    nada               ; não encontrou
            mov    ax,[esi-2]         ; valor pretendido
            jmp    tratar_valor
nada:       ...
```

Exemplo: Expressões condicionais com E-lógico (1/2)

```
if (al > bl) AND (bl > cl)
          X = 1
```

Transformação direta:

```
            cmp     al,bl       ; a1>b1
            ja      L1
            jmp     next
L1:         cmp     bl,cl       ; b1>c1
            ja      L2
            jmp     next
L2:         mov     X,1         ; ambas verdadeiras
next:       ...                 ; pelo menos uma falsa
```


Exemplo: Expressões condicionais com E-lógico (2/2)

```
if (al > bl) AND (bl > cl)
          X = 1
```

Versão mais curta complementa os testes:

```
            cmp     al,bl       ; 1º teste
            jbe     next        ; terminar se falso
            cmp     bl,cl       ; 2º teste
            jbe     next        ; terminar se falso
            mov     X,1         ; ambos verdadeiros
next:       ...
```

Expressões condicionais com OU-lógico

```
if (al > bl) OR (bl > cl)
          X = 1
```

Versão curta complementa os testes:

```
            cmp     al,bl    ; AL > BL?
            ja      L1       ; sim
            cmp     bl,cl    ; não:  e BL > CL?
            jbe     next     ; não:  evitar próxima instrução
L1:         mov     X,1
next:       ...              ; ambas falsas
```


Ciclos WHILE

```
while (eax < ebx)
          eax = eax + 1
```

Ciclo `while`:

1. instrução IF: salto para fim se condição for falsa
2. corpo do ciclo
3. salto incondicional para início

```
        topo:     cmp     eax,ebx   ; verificar condição
                  jae     next      ; falsa?  terminar
                  inc     eax       ; corpo do ciclo
                  jmp     topo      ; repetir
        next:     ...
```

Geração automática de expressões condicionais

- O código *assembly* para expressões condicionais complexas pode ser bastante complicado.
- MASM pode gerar automaticamente código para expressões condicionais de testes e ciclos
- As diretivas `.IF`, `.ELSE`, `.ELSEIF`, e `.ENDIF` são usadas para descrever testes "estruturados".

```
.IF   eax > ebx
mov   edx,1
.ELSE
mov   edx,2
.ENDIF
```

```
.IF   eax > ebx && eax > ecx
mov   edx,1
.ELSE
mov   edx,2
.ENDIF
```

➠ Apenas usar testes correspondentes a instruções de comparação válidas.

Operadores lógicos e relacionais

Operator	Description
expr1 == *expr2*	Returns true when *expression1* is equal to *expr2*.
expr1 != *expr2*	Returns true when *expr1* is not equal to *expr2*.
expr1 > *expr2*	Returns true when *expr1* is greater than *expr2*.
expr1 >= *expr2*	Returns true when *expr1* is greater than or equal to *expr2*.
expr1 < *expr2*	Returns true when *expr1* is less than *expr2*.
expr1 <= *expr2*	Returns true when *expr1* is less than or equal to *expr2*.
! *expr*	Returns true when *expr* is false.
expr1 && *expr2*	Performs logical AND between *expr1* and *expr2*.
expr1 || *expr2*	Performs logical OR between *expr1* and *expr2*.
expr1 & *expr2*	Performs bitwise AND between *expr1* and *expr2*.
CARRY?	Returns true if the Carry flag is set.
OVERFLOW?	Returns true if the Overflow flag is set.
PARITY?	Returns true if the Parity flag is set.
SIGN?	Returns true if the Sign flag is set.
ZERO?	Returns true if the Zero flag is set.

Comparações sem sinal

- MASM guia-se pelo tipo dos dados para gerar os testes

```
     .data
val1 DWORD 5
res  DWORD ?
     .code
     mov   eax, 6
     .IF   eax > val1
     mov   res, 1
     .ENDIF
```

⇒

```
     mov   eax, 6
     cmp   eax, val1
     jbe   @C0001
     mov   res, 1
@C0001:
```

- Como `val1` é *unsigned*, MASM usa o salto JBE.

Comparações com sinal

```
     .data
val1 SDWORD 5
res  SDWORD ?
     .code
     mov   eax, 6
     .IF   eax > val1
     mov   res, 1
     .ENDIF
```

⇒

```
     mov   eax, 6
     cmp   eax, val1
     jle   @C0001
     mov   res, 1
@C0001:
```

- MASM gera automaticamente salto "com sinal" (JLE), porque `val1` representa uma grandeza com sinal *(signed)*.

Comparações entre registos (1/2)

- Quando ambos os operandos são registos, MASM gera um teste *unsigned*.

```
        .data
res     DWORD ?
        .code
        mov     ebx, 5
        mov     eax, 6
        .IF     eax > ebx
        mov     res, 1
        .ENDIF
```

⇒

```
        mov     ebx, 5
        mov     eax, 6
        cmp     eax, ebx
        jbe     @C0001
        mov     res,1
@C0001:
```


Comparações entre registos (2/2)

- Para se obter uma comparação do tipo *signed*, é necessário aplicar um atributo de tipo a um dos registos.

```
        .data
res     BYTE ?
        .code
        mov     ebx, 5
        mov     eax, -1
        .IF     SDWORD PTR eax > ebx
        mov     res, 1
        .ENDIF
```

⇒

```
        mov     ebx, 5
        mov     eax, 6
        cmp     eax, ebx
        jle     @C0001
        mov     res,1
@C0001:
```

As diretivas .REPEAT e .WHILE

- A diretiva .REPEAT repete um ciclo até o teste ser verdadeiro.

```
mov       ebx, 0
.REPEAT
inc       ebx
;; fazer alguma coisa
.UNTIL    ebx == 10
```

- A diretiva .WHILE repete um ciclo enquanto o teste for verdadeiro.

```
mov       ebx, 0
.WHILE    ebx < 10
inc       ebx
;; fazer alguma coisa
.ENDW
```