

Assembly Language for Intel-Based Computers, 5th Edition

Kip Irvine

Capítulo 4 – Parte A

Acesso à Memória, Transferência de Dados, Instruções Aritméticas

Slides prepared by the author

Revision date: June 4, 2006

(c) Pearson Education, 2006-2007. All rights reserved. You may modify and copy this slide show for your personal use, or for use in the classroom, as long as this copyright statement, the author's name, and the title are not changed.

Instrução de Transferência de Dados

- Tipos de Operandos
- Notação de Operandos de Instrução
- Operandos de Memória por endereçamento Direto
- Instrução MOV
- Extensão de Zero & Sign
- Instrução XCHG
- Operandos com Deslocamento Direto (Direct-Offset)

Tipos de Operandos

- Três tipos básicos de operandos:
 - Imediato – uma constante inteira (8, 16, ou 32 bits)
 - Valor é codificado dentro da instrução
 - Registrador – nome de um register
 - O nome do registrador é codificado dentro da instrução
 - Memória – referência a uma posição na memória
 - O endereço da memória é codificado dentro da instrução, ou um registrador contém o endereço de uma posição de memória

Notação de Operandos de Instrução

Operand	Description
<i>r8</i>	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
<i>r16</i>	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
<i>r32</i>	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
<i>reg</i>	any general-purpose register
<i>sreg</i>	16-bit segment register: CS, DS, SS, ES, FS, GS
<i>imm</i>	8-, 16-, or 32-bit immediate value
<i>imm8</i>	8-bit immediate byte value
<i>imm16</i>	16-bit immediate word value
<i>imm32</i>	32-bit immediate doubleword value
<i>r/m8</i>	8-bit operand which can be an 8-bit general register or memory byte
<i>r/m16</i>	16-bit operand which can be a 16-bit general register or memory word
<i>r/m32</i>	32-bit operand which can be a 32-bit general register or memory doubleword
<i>mem</i>	an 8-, 16-, or 32-bit memory operand

Operandos de Memória por Endereçamento Direto

- Um operando de memória por endereçamento direto é uma referência à memória por nome
- A referência (label) é automaticamente reconhecida pelo assembler

```
.data  
var1 BYTE 10h  
.code  
mov al,var1           ; AL = 10h  
mov al,[var1]         ; AL = 10h
```

alternativa



Instrução MOV

- Move da fonte para o destino. Sintaxe:
MOV destino, fonte
- É permitido apenas um operando de memória na instrução mov
- CS, EIP, e IP não podem ser destino
- Não existe mov imediato para registrador de segmento

```
.data
count BYTE 100
wVal  WORD 2
.code
    mov bl, count
    mov ax, wVal
    mov count, al

    mov al, wVal           ; error
    mov ax, count          ; error
    mov eax, count         ; error
```

Sua vez . . .

Explicar por que as instruções MOV seguintes são inválidas:

```
.data
bVal  BYTE    100
bVal2 BYTE    ?
wVal  WORD     2
dVal  DWORD    5
.code
    mov ds,45

    mov esi,wVal

    mov eip,dVal

    mov 25,bVal

    mov bVal2,bVal
```

Sua vez . . .

Explicar por que as instruções MOV seguintes são inválidas:

```
.data
bVal  BYTE    100
bVal2 BYTE     ?
wVal  WORD     2
dVal  DWORD    5
.code
    mov ds,45          Move imediato para DS não permitido

    mov esi,wVal       Tamanho incompatível

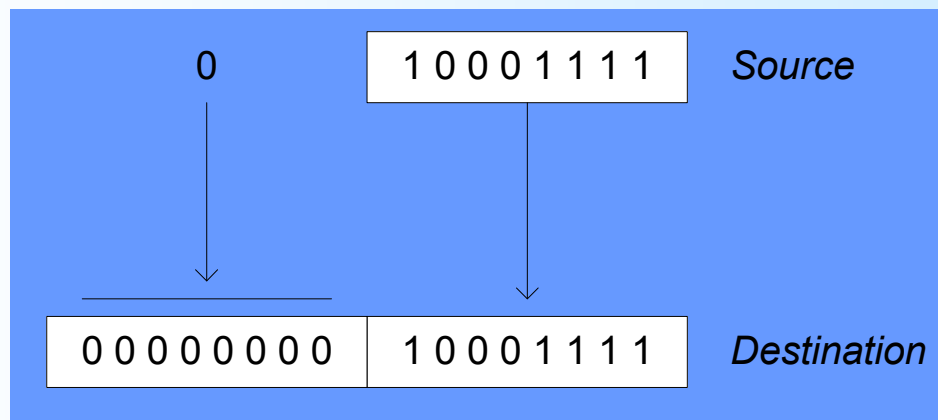
    mov eip,dVal       EIP não pode ser destino

    mov 25,bVal        Valor imediato não pode ser destino

    mov bVal2,bVal     Move memória-a-memória não permitido
```


Extensão com Zeros

Quando se copia um valor menor para um destino maior, a instrução MOVZX estende a parte mais significativa do destino com zeros.

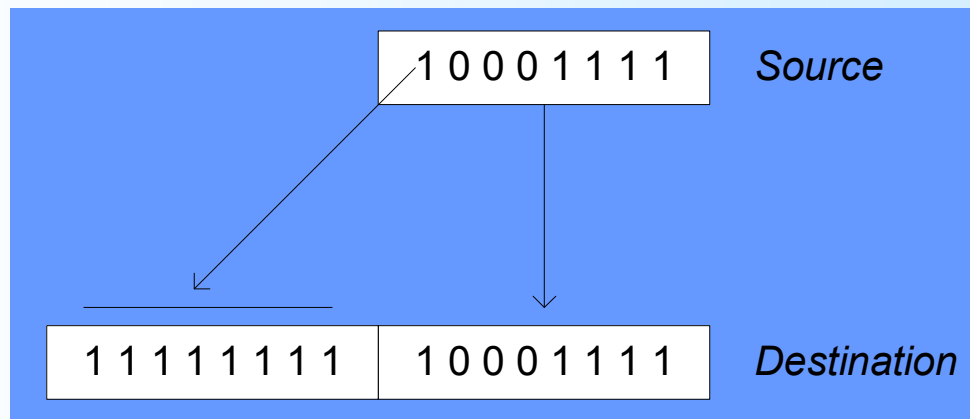


```
mov bl,10001111b  
movzx ax,bl ; zero-extension
```

O destino deve ser um registrador.

Extensão com Sinal

A instrução MOVSX preenche a parte mais significativa do destino com uma cópia do bit de sinal do operando fonte.



```
mov bl,10001111b  
movsx ax,bl           ; sign extension
```

O destino deve ser um registrador.

Instrução XCHG

XCHG troca os valores de dois operandos. Pelo menos um operando deve ser um registrador. Nenhum operando imediato é permitido.

```
.data
var1 WORD 1000h
var2 WORD 2000h
.code
xchg ax,bx           ; exchange 16-bit regs
xchg ah,al           ; exchange 8-bit regs
xchg var1,bx         ; exchange mem, reg
xchg eax,ebx         ; exchange 32-bit regs

xchg var1,var2       ; error: two memory operands
```

Operandos com deslocamento direto (Direct-Offset)

Um deslocamento é adicionado a um label para produzir um endereço efetivo.

O endereço é calculado pelo Assembler.

```
.data
arrayB BYTE 10h,20h,30h,40h
.code
mov al,arrayB+1           ; AL = 20h
mov al,[arrayB+1]         ; alternative notation
```

Por que arrayB+1 não resulta em 11h?

Operandos com deslocamento direto (Direct-Offset)(cont)

```
.data
arrayW  WORD 1000h,2000h,3000h
arrayD  DWORD 1,2,3,4
.code
mov ax,[arrayW+2]           ; AX = 2000h
mov ax,[arrayW+4]           ; AX = 3000h
mov eax,[arrayD+4]          ; EAX = 00000002h
```

```
; As seguintes instruções são corretas?
mov ax,[arrayW-2]           ; ??
mov eax,[arrayD+16]         ; ??
```

O que acontece quando são executadas?

Sua vez. . .

Escrever um programa que rearranja os valores de 3 doublewords no seguinte vetor: 3, 1, 2.

```
.data  
arrayD DWORD 1,2,3
```

Sua vez. . .

Escrever um programa que rearranja os valores de 3 doublewords no seguinte vetor: 3, 1, 2.

```
.data  
arrayD DWORD 1,2,3
```

- passo1: copiar o primeiro valor em EAX e trocar com o valor na segunda posição.

```
mov eax,arrayD  
xchg eax,[arrayD+4]
```

- passo 2: trocar EAX com o terceiro valor e copiar o valor em EAX para a primeira posição.

```
xchg eax,[arrayD+8]  
mov  arrayD,eax
```

Avaliar (1)

- Queremos escrever um programa que soma os 3 bytes seguintes:

```
.data  
myBytes BYTE 80h,66h,0A5h
```


Avaliar (1)

- Queremos escrever um programa que soma os 3 bytes seguintes:

```
.data  
myBytes BYTE 80h,66h,0A5h
```

- Qual a avaliação para o seguinte código?

```
mov al,myBytes  
add al,[myBytes+1]  
add al,[myBytes+2]
```

- Qual a avaliação para o seguinte código?

```
mov ax,myBytes  
add ax,[myBytes+1]  
add ax,[myBytes+2]
```

- Alguma outra possibilidade?

Avaliar (1)

```
.data  
myBytes BYTE 80h,66h,0A5h
```

- Falta algo no código seguinte?

```
movzx ax,myBytes  
mov    bl,[myBytes+1]  
add    ax,bx  
mov    bl,[myBytes+2]  
add    ax,bx  
; AX = sum
```

Avaliar (1)

```
.data  
myBytes BYTE 80h,66h,0A5h
```

- Falta algo no código seguinte?

```
movzx ax,myBytes  
mov    bl,[myBytes+1]  
add    ax,bx  
mov    bl,[myBytes+2]  
add    ax,bx                ; AX = sum
```

Sim: **Mover zero a BX antes da instrução MOVZX.**

Adição e Subtração

- Instruções INC e DEC
- Instruções ADD e SUB
- Instrução NEG
- Implementando Expressões Aritméticas
- Flags afetados pela Aritmética
 - Zero
 - Sign
 - Carry
 - Overflow

Instruções INC e DEC

- INC *destino*
 - Lógica: $\text{destino} \leftarrow \text{destino} + 1$
- DEC *destino*
 - Lógica: $\text{destino} \leftarrow \text{destino} - 1$

Exemplos de INC e DEC

```
.data
myWord  WORD 1000h
myDword DWORD 10000000h
.code
    inc myWord           ; 1001h
    dec myWord           ; 1000h
    inc myDword          ; 10000001h

    mov ax,00FFh
    inc ax               ; AX = 0100h
    mov ax,00FFh
    inc al               ; AX = 0000h
```

Sua vez...

Mostrar o valor do operando destino depois que cada instrução é executada:

```
.data
myByte BYTE 0FFh, 0
.code
    mov al,myByte           ; AL =
    mov ah,[myByte+1]      ; AH =
    dec ah                  ; AH =
    inc al                  ; AL =
    dec ax                  ; AX =
```

Sua vez...

Mostrar o valor do operando destino depois que cada instrução é executada:

```
.data
myByte BYTE 0FFh, 0
.code
    mov al,myByte           ; AL = FFh
    mov ah,[myByte+1]       ; AH = 00h
    dec ah                  ; AH = FFh
    inc al                   ; AL = 00h
    dec ax                   ; AX = FFFFh
```


Instruções ADD e SUB

- ADD destino, fonte
 - Lógica: $\text{destino} \leftarrow \text{destino} + \text{fonte}$
- SUB destino, fonte
 - Lógica: $\text{destino} \leftarrow \text{destino} - \text{fonte}$
- Obs: Mesmas regras de operando que a instrução MOV

Exemplos de ADD e SUB

```
.data
var1 DWORD 10000h
var2 DWORD 20000h
.code
; ---EAX---
mov eax,var1      ; 00010000h
add eax,var2      ; 00030000h
add ax,0FFFFh     ; 0003FFFFh
add eax,1         ; 00040000h
sub ax,1          ; 0004FFFFh
```

Instrução NEG

Nega um operando. O operando pode ser de registrador ou memória.

```
.data
valB BYTE -1
valW WORD +32767
.code
    mov al, valB           ; AL = -1
    neg al                 ; AL = +1
    neg valW               ; valW = -32767
```

Supondo que AX contenha $-32,768$, aplicar NEG.

O resultado é válido?

Instrução NEG e Flags

O processador implementa NEG usando a seguinte operação interna:

zero menos operando (0 - operando)

Qualquer operando diferente de zero faz com que o Carry flag seja acionado.

```
.data
valB BYTE 1,0
valC SBYTE -128
.code
    neg valB                ; CF = 1, OF = 0
    neg [valB + 1]          ; CF = 0, OF = 0
    neg valC                ; CF = 1, OF = 1
```

Implementando Expressões Aritméticas

Compiladores traduzem expressões matemáticas em linguagem Assembly, o que equivale ao processo manual abaixo:

Ex: `Rval = -Xval + (Yval - Zval)`

```
Rval DWORD ?
Xval  DWORD 26
Yval  DWORD 30
Zval  DWORD 40
.code
    mov eax,Xval
    neg eax                ; EAX = -26
    mov ebx,Yval
    sub ebx,Zval           ; EBX = -10
    add eax,ebx
    mov Rval,eax           ; -36
```

Sua vez...

Traduzir a seguinte expressão em linguagem Assembly. Não modificar Xval, Yval, e Zval:

$$\text{Rval} = \text{Xval} - (-\text{Yval} + \text{Zval})$$

Assumir que todos os valores são doublewords com sinal.

Sua vez...

Traduzir a seguinte expressão em linguagem Assembly. Não modificar Xval, Yval, e Zval:

$$Rval = Xval - (-Yval + Zval)$$

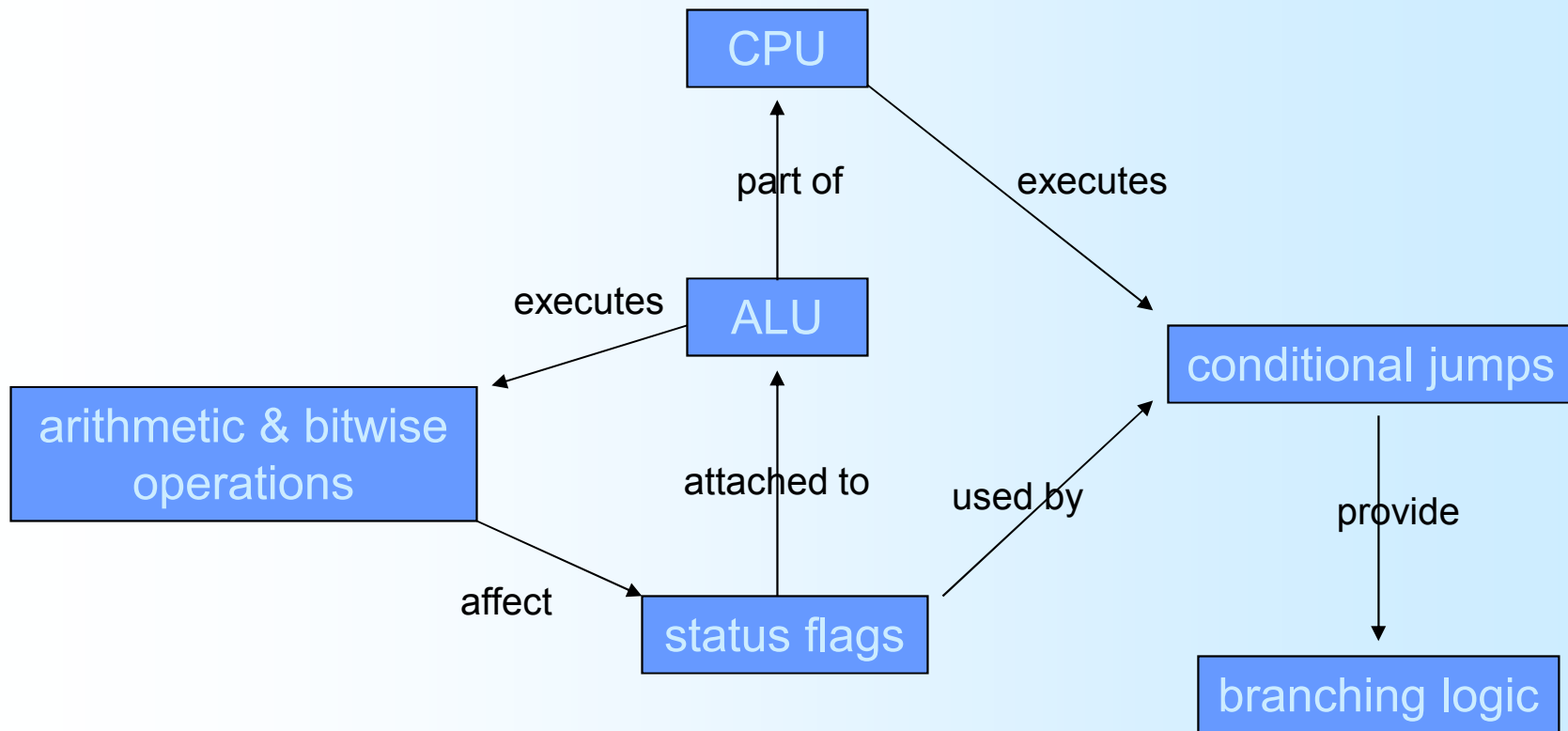
Assumir que todos os valores são doublewords com sinal.

```
mov ebx,Yval  
neg ebx  
add ebx,Zval  
mov eax,Xval  
sub eax,ebx  
mov Rval,eax
```

Flags afetados pela aritmética

- A ALU tem um número de flags que refletem o resultado da aritmética e operações com bits;
- Baseado no conteúdo do operando destino;
- Flags essenciais:
 - Zero flag – aciona quando o destino é zero
 - Sign flag – aciona quando o destino é negativo
 - Carry flag – aciona quando um valor sem sinal cair fora do intervalo
 - Overflow flag – aciona quando um valor com sinal cair fora do intervalo
- A instrução MOV não afeta as flags.

Diagrama ilustrativo



Pode-se usar diagramas como esse para expressar as relações entre os conceitos da linguagem assembly.

Flag Zero (ZF)

O flag Zero é acionado quando o resultado de uma operação produz zero no operando destino.

```
mov cx,1
sub cx,1           ; CX = 0, ZF = 1
mov ax,0FFFFh
inc ax             ; AX = 0, ZF = 1
inc ax             ; AX = 1, ZF = 0
```

Lembrar ...

- Acionar flag é torná-lo igual a 1.
- Zerar flag é torná-lo igual a 0.

Flag Sign (SF)

O flag Sign é acionado quando o operando destino é negativo.
O flag é zerado quando o operando destino é positivo.

```
mov cx,0
sub cx,1           ; CX = -1, SF = 1
add cx,2           ; CX = 1, SF = 0
```

O sign flag é uma cópia do bit de sinal do operando destino

```
mov al,0
sub al,1           ; AL = 11111111b, SF = 1
add al,2           ; AL = 00000001b, SF = 0
```

Inteiros com sinal e sem sinal

Ponto de vista do Hardware

- Todas as instruções operam exatamente da mesma forma em inteiros com sinal e sem sinal
- A CPU não faz distinção entre números inteiros com sinal e sem sinal
- O programador é o único responsável pelo uso correto dos tipos de dados para cada instrução

Flags de Overflow e Carry

Ponto de vista do Hardware

- Como a instrução ADD modifica OF e CF:
 - $OF = (\text{carry out do MSB}) \text{ xor } (\text{carry in para MSB})$
 - $CF = (\text{carry out of the MSB})$
- Como a instrução SUB modifica OF e CF:
 - Nega (NEG) o fonte e soma (ADD) ao destino
 - $OF = (\text{carry out do MSB}) \text{ xor } (\text{carry in para MSB})$
 - $CF = \text{INVERT} (\text{carry out do MSB})$

MSB = Bit Mais Significante (high-order bit)

XOR = OU Exclusivo

NEG = Negate (equivalente a `SUB 0,operand`)

Added Slide. Gerald Cahill, Antelope Valley College

Irvine, Kip R. Assembly Language for Intel-Based Computers 5/e, 2007.

[Web site](#)

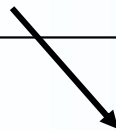
[Examples](#)

Flag Carry (CF)

O flag Carry é acionado quando o resultado de uma operação gera um **valor sem sinal** que é **fora do intervalo** (muito grande ou muito pequeno para o operando destino).

```
mov al,0FFh
add al,1                ; CF = 1, AL = 00

mov al,0
sub al,1                ; CF = 1, AL = FF
```



Resposta= 1111 1111, mas estamos tratando de um número sem sinal, logo isto não representa a resposta correta (-1), mas 255 -> CARRY = 1

Sua vez ...

Para cada uma das instruções aritméticas seguintes, mostrar os valores do operando destino e os flags de Sign, Zero e Carry :

<code>mov ax,00FFh</code>				
<code>add ax,1</code>	<code>; AX=</code>	<code>SF=</code>	<code>ZF=</code>	<code>CF=</code>
<code>sub ax,1</code>	<code>; AX=</code>	<code>SF=</code>	<code>ZF=</code>	<code>CF=</code>
<code>add al,1</code>	<code>; AL=</code>	<code>SF=</code>	<code>ZF=</code>	<code>CF=</code>
<code>mov bh,6Ch</code>				
<code>add bh,95h</code>	<code>; BH=</code>	<code>SF=</code>	<code>ZF=</code>	<code>CF=</code>
<code>mov al,2</code>				
<code>sub al,3</code>	<code>; AL=</code>	<code>SF=</code>	<code>ZF=</code>	<code>CF=</code>

Sua vez ...

Para cada uma das instruções aritméticas seguintes, mostrar os valores do operando destino e os flags de Sign, Zero e Carry :

<code>mov ax,00FFh</code>	
<code>add ax,1</code>	; AX= 0100h SF= 0 ZF= 0 CF= 0
<code>sub ax,1</code>	; AX= 00FFh SF= 0 ZF= 0 CF= 0
<code>add al,1</code>	; AL= 00h SF= 0 ZF= 1 CF= 1
<code>mov bh,6Ch</code>	
<code>add bh,95h</code>	; BH= 01h SF= 0 ZF= 0 CF= 1
 <code>mov al,2</code>	
<code>sub al,3</code>	; AL= FFh SF= 1 ZF= 0 CF= 1

$6Ch + 95h = 108d + 149d = 257d = 101h$

$02h - 03h = 02h + 1D = 1111\ 1111 = 255$, e não -1

Flag Overflow (OF)

O flag Overflow é acionado quando o resultado com sinal de uma operação é inválido ou fora do intervalo.

```
; Example 1
mov al,+127
add al,1                      ; OF = 1,    AL = ??

; Example 2
mov al,7Fh                    ; OF = 1,    AL = 80h
add al,1
```

Os dois exemplos são idênticos no binário pois 7Fh é igual a +127. Para determinar o valor do operando destino, em hexadecimal é geralmente mais fácil.

Uma regra simples

- Ao somar dois inteiros, lembrar que o flag Overflow é apenas acionado quando . . .
 - Dois operandos positivos são somados e a sua soma é negativa
 - Dois operandos negativos são somados e sua soma é positiva

Qual é o valor do flag Overflow?

```
mov al,80h
add al,92h                ; OF = 1
```

```
mov al,-2
add al,+127               ; OF = 0
```

Sua vez. . .

Quais seriam os valores dos flags para cada operação?

```
mov al,-128
neg al                ; CF =      OF =

mov ax,8000h
add ax,2              ; CF =      OF =

mov ax,0
sub ax,2              ; CF =      OF =

mov al,-5
sub al,+125           ; OF =
```

Sua vez. . .


Quais seriam os valores dos flags para cada operação?

```
mov al,-128
neg al                ; CF = 1    OF = 1

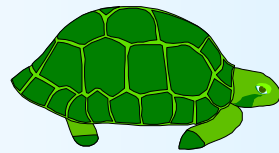
mov ax,8000h
add ax,2              ; CF = 0    OF = 0

mov ax,0
sub ax,2              ; CF = 1    OF = 0

mov al,-5
sub al,+125           ; OF = 1
```



80h -> 0 - 80h = 0 + 80h = 80h = -128d -> estouro -> Carry



The End