Linguagens de Montagem

DEMAC – Departamento de Estatística
Matemática Aplicada e Computação
UNESP – Rio Claro

Prof. Daniel Carlos Guimarães Pedronette

Aula 3.

Instruções Aritméticas e Flags

Flags de Status

- Há um conjunto de Flags com o objetivo de "monitorar" o resultado das instruções lógico-aritméticas
- Quando uma operação lógico-aritmética é executada as flags são atualizadas para indicar propriedades do resultado da operação realizada
- Cada instrução afeta um conjunto diferente de flags

Flags de Status

- Além de monitoramento, quais outras utilidades das flags:
  - Instruções de desvio condicional utilizam o valor das flags para determinar se o desvio deve ou não ser executado

Flags de Status

- Zero flag (ZF)
- Carry flag (CF)
- Overflow flag (OF)
- Sign flag (SF),
- Auxiliary flag (AF)
- Parity flag (PF)

Zero Flag (ZF)

- Indica que o resultado da última instrução produziu um resultado igual a zero
  - Muito intuitivo para subtrações por exemplo
  - Mas não para todos os casos. No exemplo abaixo ZF=1 pois AL fica com todos os bits igual a zero

```
mov     AL,0FH
add     AL,0F1H
```

Exemplo: Flag update

- Zero Flag:

```
;initially, assume that ZF is 0
mov     EAX,55H   ; ZF is still 0
sub     EAX,55H   ; result is zero
                  ; Thus, ZF is set (ZF = 1)
push    EBX       ; ZF remains 1
mov     EBX,EAX   ; ZF remains 1
pop     EDX       ; ZF remains 1
mov     ECX,0     ; ZF remains 1
inc     ECX       ; result is 1
                  ; Thus, ZF is cleared (ZF = 0)
```

Carry Flag (CF)

- Indica que o resultado da última instrução está fora do "range" (muito pequeno ou muito grande) para números sem sinal

```
   mov     AL,0FH
   add     AL,0F1H

   00001111B   (0FH = 15D)
   11110001B   (F1H = 241D)
 1 00000000B   (100H = 256D)
```

```
mov     EAX,12AEH
sub     EAX,12AFH
```

CF=1

Carry Flag

- Intervalos para números sem sinal:

Size (bits)	Range
8	0 to 255
16	0 to 65,535
32	0 to 4,294,967,295

Overflow Flag (OF)

- Análogo ao Carry para números com sinal. Indica que o resultado da última instrução está fora do "range" suportado.
  - Exemplo: OF=1 porque o resultado (80H, 128D é muito grande em 8 bits com sinal)

```
mov     AL,72H   ; 72H = 114D
add     AL,0EH   ; 0EH = 14D
```

Overflow Flag

- Intervalos para números com sinal:

Size (bits)	Range
8	−128 to +127
16	−32,768 to +32,767
32	−2,147,483,648 to +2,147,483,647

Com sinal ou sem sinal?

- Uma posição de memória e seu padrão de bits pode ser utilizado para representar números com sinal e sem sinal
- Se os flags de carry e overflow dependem do sinal, como o processador diferencia um número com ou sem sinal?
  - Ele não diferencia, isso é papel da lógica do programa.

Com sinal ou sem sinal?

- O processador considera ambas as situações (com e sem sinal) e define as flags como necessário.
  - Exemplo:

```
mov     AL,72H
add     AL,0EH
```

Sem sinal (80h, 128 < 255), CF=0

Com sinal (80h, 128 > 127), OF=1

Sign Flag

- Indica o sinal do resultado de uma operação aritmética
  - Bit mais significativo é utilizado para representar o sinal de um número
    - 0 para números positivos
    - 1 para números negativos
  - Números negativos são representados em complemento de 2
- Sign Flag armazena uma cópia desse bit do resultado produzido pelas instruções aritméticas

Sign Flag

- Exemplos:

```
mov     EAX,15
add     EAX,97
```

SF=0

```
mov     EAX,15
sub     EAX,97
```

SF=1

```
  00001111B     (8-bit signed form of 15)
+ 10011111B     (8-bit signed number for −97)
  ----------
  10101110B
```

SF=1

Instruções Lógico-Aritméticas

- INC, DEC
- ADD, SUB
- CMP
- AND, OR
- XOR
- NOT
- MUL
-

Instruções INC e DEC

- INCremento e DECremento

```
inc     destination
dec     destination
```

```
inc     EBX         ; increment 32-bit register
dec     DL          ; decrement 8-bit register
```

EBX and DL	1057H and 5AH	FFFFH and 00H
	1058H and 59H	0000H and FFH

Instrução ADD

- Soma *dest* + *source* e armazena o resultado em *dest*

```
add     destination,source

destination = destination + source
```

Instruction	Before add		After add
	Source	Destination	Destination
add AX,DX	DX = AB62H	AX = 1052H	AX = BBB4H
add BL,CH	BL = 76H	CH = 27H	BL = 9DH
add value,10H	—	value = F0H	value = 00H
add DX,count	count = 3746H	DX = C8B9H	DX = FFFFH

Instrução SUB

- Subtrai *source* de *dest* e armazena o resultado em *dest*

```
sub     destination,source

destination = destination – source
```

Instruction	Before sub		After sub
	Source	Destination	Destination
`sub AX,DX`	`DX = AB62H`	`AX = 1052H`	`AX = 64F0H`
`sub BL,CH`	`CH = 27H`	`BL = 76H`	`BL = 4FH`
`sub value,10H`	—	`value = F0H`	`value = E0H`
`sub DX,count`	`count = 3746H`	`DX = C8B9H`	`DX = 9173H`

Instrução CMP

- Utilizada para comparar dois operandos
- Executa a mesma operação da instrução SUB
  - A diferença consiste no fato de que a instrução CMP são “salva” o resultado da subtração, não alterando os operandos
  - Utilizada em conjunto com instruções de desvio

Instruções Lógicas

- Instruções lógicas binárias

```
and     destination,source
or      destination,source
xor     destination,source
```

- Instrução lógica unária

```
not     destination
```

Instruções Lógicas

- AND
  - Source e Destination

and Operation

Input bits		Output bit
Source b_i	Destination b_i	Destination b_i
0	0	0
0	1	0
1	0	0
1	1	1

Instruções Lógicas

- OR
  - Source e Destination

`or` Operation

Input bits		Output bit
Source b_i	Destination b_i	Destination b_i
0	0	0
0	1	1
1	0	1
1	1	1

Instruções Lógicas

- XOR
  - Source e Destination

`xor` Operation

Input bits		Output bit
Source b_i	Destination b_i	Destination b_i
0	0	0
0	1	1
1	0	1
1	1	0

Instruções Lógicas

- Exemplos:

AL	BL	and AL,BL	or AL,BL	xor AL,BL	not AL
		AL	AL	AL	AL
1010 1110	1111 0000	1010 0000	1111 1110	0101 1110	0101 0001
0110 0011	1001 1100	0000 0000	1111 1111	1111 1111	1001 1100
1100 0110	0000 0011	0000 0010	1100 0111	1100 0101	0011 1001
1111 0000	0000 1111	0000 0000	1111 1111	1111 1111	0000 1111

Instrução MUL

- A instrução é um pouco mais complicada que a soma e subtração:
  - Multiplicação produz resultados *double-lenght*
    - Multiplicar dois números de 8 bits requer um *destination* de armazenamento de 16 bits

$$\underset{(255D)}{11111111} \times \underset{(255D)}{11111111} = \underset{(65025D)}{1111111011111111}.$$

Instrução MUL

- Multiplicação de números com sinal e sem sinal precisam ser tratados de maneira diferente
  - O padrão de bits depende do tipo da entrada

FFH

- Sem sinal

255D

$$11111111 \times 11111111 = 11111110\ 11111111$$

- Com sinal

−1D

$$11111111 \times 11111111 = 00000000\ 00000001$$

Instrução MUL

- Sem sinal, formato:

```
mul     source
```

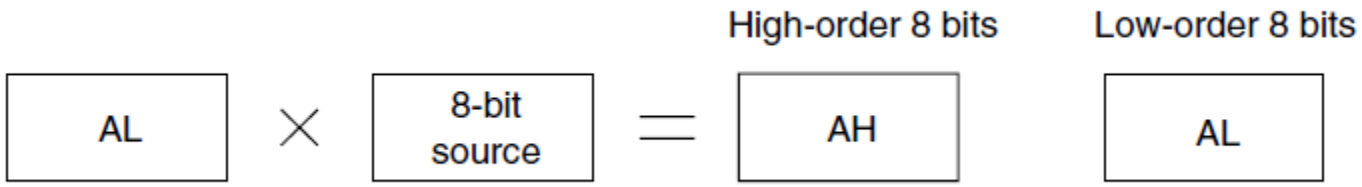
  - Onde está o segundo operando?
  - A instrução assume que está no registrador acumulador.
    - Onde será armazenado o resultado?

Instrução MUL

- Sem sinal, formato:

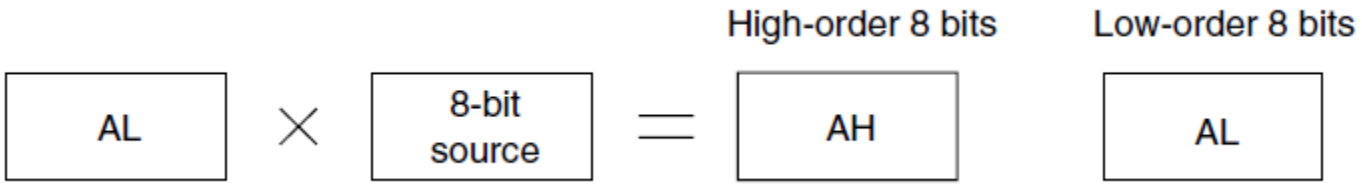
```
mul     source
```

  - Onde está o segundo operando?
  - A instrução assume que está no registrador acumulador.

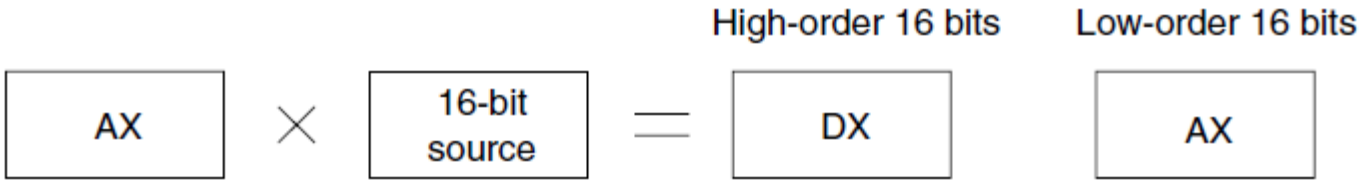


Instrução MUL

- 8 bits:

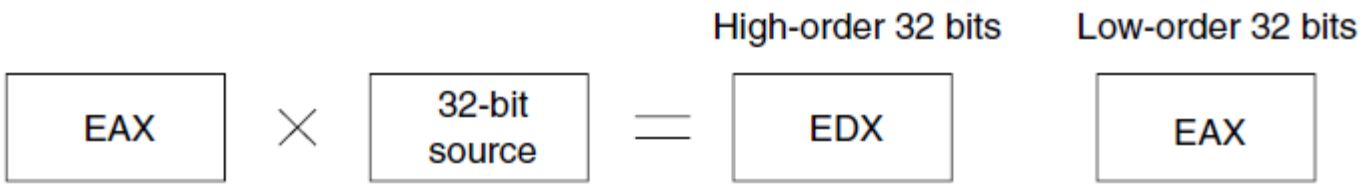


- 16 bits:



Instrução MUL

- 32 bits:



Instrução IMUL

- Multiplicação com sinal:

```
imul    source
```

```
mov     DL,0FFH     ; DL = -1
mov     AL,0BEH     ; AL = -66
imul    DL
```

0000000001000010 (+66)

```
mov     DL,25     ; DL = 25
mov     AL,0F6H ; AL = -10
imul    DL
```

1111111100000110 (−250)

Exemplos

Prática:

- Codificar,
- Montar,
- Linkar e
- Testar!

Exercícios