Laboratório de Pesquisa em Redes e Multimídia

Programando em Assembly

(Aula 16)

Linguagem Assembly do 8086/8088

Universidade Federal do Espírito Santo
Departamento de Informática

Roberta Lima Gomes - LPRM/DI/UFES
Sistemas de Programação I – Eng. Elétrica
2007/2

Introdução

- Para construirmos os programas em Assembly, devemos estruturar o fonte da seguinte forma (usando TASM como montador)

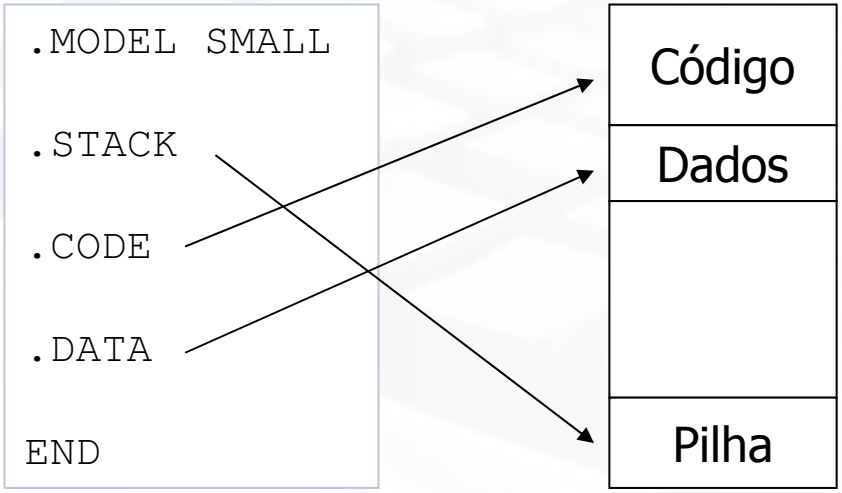
```
.MODEL SMALL
Define o modelo de memória a usar em nosso programa

.STACK
Reserva espaço de memória para as instruções de programa na pilha

.CODE
Define as instruções do programa, relacionado ao segmento de código

END
Finaliza um programa assembly
```

Primeiro Exemplo (1)

```
.MODEL SMALL ;modelo de memória
.STACK       ;espaço de memória para instruções do programa na pilha
.CODE        ;as linhas seguintes são instruções do programa
  mov ah,01h ;move o valor 01h para o registrador ah
  mov cx,07h ;move o valor 07h para o registrador cx
  int 10h    ;interrupção 10h
  mov ah,4ch ;move o valor 4ch para o registrador ah
  int 21h    ;interrupção 21h
.DATA
x db 1
END          ;finaliza o código do programa
```

Este programa assembly muda o tamanho do cursor

Primeiro Exemplo (2)



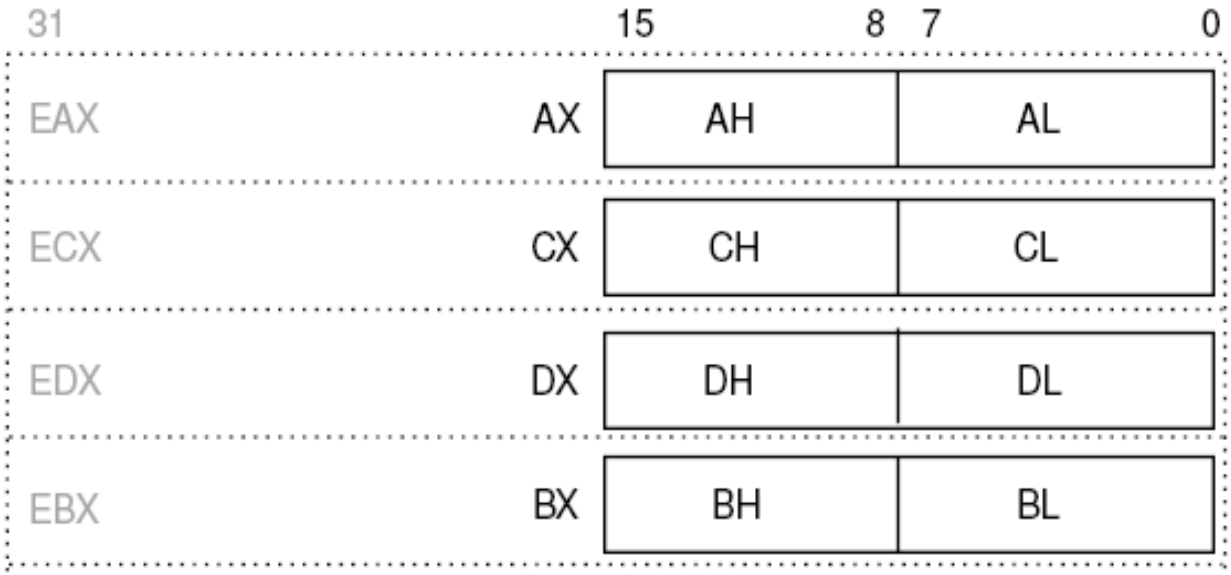
- A memória é dividida em um número arbitrário de segmentos

Registradores de Uso Geral (1)

```
AX     BX     CX     DX             (16 bits)
AH/AL  BH/BL  CH/CL  DH/DL          (8 bits)
```

- **AX:** Acumulador
  - Usado em operações aritméticas.
- **BX:** Base
  - Usado para indexar tabelas de memória (ex.: índice de vetores).
- **CX:** Contador
  - Usado como contador de repetições em loop e movimentação repetitiva de dados.
- **DX:** Dados
  - Uso geral.

Registradores de Uso Geral (2)



Acumulador

Reg. de contagem: string, loop

Reg. de dados: multiplicação, divisão

Reg. de endereço básico

Usando Instruções de transferência de dados

- **`MOV Destino,Fonte`**

Operação		Exemplo
registrador,	registrador	mov Bx, Cx
memória,	acumulador	mov var, Al
acumulador,	memória	mov Ax, var
memória,	registrador	mov var, Si
registrador,	memória	mov Si, var
registrador,	imediato	mov var, 12
reg_seg,	reg16	mov Ds, Ax
reg16, reg_seg		mov Ax, Ds
memória,	reg_seg	mov var, Ds

Exemplo 2

- Faça um programa em ASM86 correspondente ao seguinte "código C" (use "mov" e "dw")

```
unsigned int x, y, z;
main()
{
    x = 7;
    y = 13;
    z = x;
}
```

```
.MODEL SMALL
.STACK
.CODE
        mov x,7
        mov y,13
        mov DX,x
        mov z,DX
.DATA
x       dw      ?
y       dw      ?
z       dw      ?
END
```

Usando Instruções Aritméticas (1)

- **`ADD destino,fonte`**

`(destino ← destino+origem)`

Operação	Exemplo
registrador, registrador	Add Dx, Bx
registrador, memória	Add Bx, var
memória, registrador	Add var, Si
acumulador,imediato	Add Al, 5
registrador,imediato	Add Dx, 3
memória, imediato	Add var, 0Fh

- **`SUB destino,fonte`**

`(destino ← destino-origem)`

Operação	Exemplo
registrador, registrador	Sub Dx, Bx
registrador, memória	Sub Bx, var
memória, registrador	Sub var, Si
acumulador, imediato	Sub Al, 5
registrador, imediato	Sub Dx, 3
memória, imediato	Sub var, 0Fh

Exemplo 3

- Faça um programa em ASM86 correspondente ao seguinte "código C" (use "mov","dw")

```
unsigned int x = 5;
unsigned int y = 10;
unsigned int soma, sub;
main()
{
    x = x + 8;
    soma = x + y;
    x = x - 3 ;
    sub = x - y
}
```

```
.MODEL SMALL
.STACK
.CODE
        add x,8          ;x=x+8
        mov DX,x         ;DX=x
        add DX,y         ;DX=DX+y=x+y
        mov soma,Dx      ;soma=DX=x+y
        sub x,3 ;        ;x=x-3
        mov DX,x         ;DX=x
        sub DX,y         ;DX=DX-y=x-y
        mov sub,DX       ;sub=DX=x-y
.DATA
x       dw      5
y       dw      10
soma    dw      ?
sub     dw      ?
END
```

Usando Instruções Aritméticas (2)

- **`MUL origem`**

  Se (origem == byte)
  AX = AL * origem
  Se (origem == word)
  DX:AX = AX * origem

Operando	Exemplo
imediato	MUL 20
reg8	MUL CH
reg16	MUL BX
mem8	MUL VarByte
mem16	MUL VarWord

- **`DIV origem`**

  Se (origem == byte)
  AL = AX / origem
  AH = resto
  Se (origem == ord)
  AX = DX:AX / origem
  DX = resto

Operando	Exemplo
imediato	DIV 16
reg8	DIV BL
reg16	DIV BX
mem8	DIV VarByte
mem16	DIV VarWord

Exemplo 4

- Faça um programa em ASM86 correspondente ao seguinte "código C" (use "mov","dw")

```
unsigned int x = 5;
unsigned int y = 12;
unsigned int sqr,dv, rest;
main()
{
   sqr = x * x
   dv = x / y;
   rest = x % y;
}
```

```
.MODEL SMALL
.STACK
.CODE
        mov AL,x        ;AL=x
        mul x           ;AX=AL*x = x*x
        mov sqr,AX      ;sqr=AX = x*x
        mov AL,x        ;AL=x
        mov AH,0        ;AX<->[AH,AL]
        div y           ;AL=AX/y
                        ;AH=Resto(AX/y)
        mov dv,AL       ;dv=AL
        mov rest,AH     ;rest=AH
.DATA
x       db      5
y       db      12
sqr     dw      ?
dv      db      ?
rest    db      ?
END
```

Usando Instruções Lógicas

- **`XOR destino,fonte`**

  `destino ← destino xor origem (bit a bit)`

Operação	Exemplo
registrador, registrador	Xor Bx, Cx
registrador, memória	Xor Cl, var
memória, registrador	Xor var, Dx
acumulador, imediato	Xor Al, 10101111b
registrador, imediato	Xor Si, 0CAh
memória, imediato	Xor var, 3

- **`AND, OR`**
- **`NOT`** (sintaxe só c/ **`destino`**)

Exemplo 5

- Implemente um trecho de código em Assembly que troque o valor do registrador AH com o valor do registrador BH
  - Utilize apenas instruções lógicas XOR
  - Você pode utilizar outros registradores para guardar valores intermediários

```
xor  DH,DH        ;DH=0
xor  DH,AH        ;DH=AH
xor  AH,AH        ;AH=0
xor  AH,BH        ;AH=BH
xor  BH,BH        ;BH=0
xor  BH,DH        ;BH=DH
```

Instruções usadas no controle de fluxo (1)

- **`CMP destino, origem`**

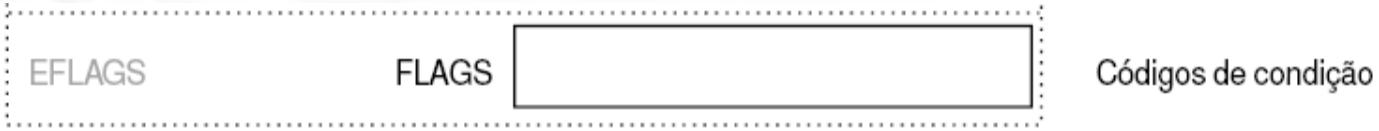
Esta instrução subtrai o operador origem do destino (destino – origem), mas não armazena o resultado da operação, apenas afeta o estado das flags de estado

Operação	Exemplo
Registrador, registrador	Cmp Cx, Bx
Registrador, imediato	Cmp Si, 3
Acumulador, imediato	Cmp Al, 0Fh
Registrador, memória	Cmp Bx, var
Memória, registrador	Cmp var, Cx
Memória, imediato	Cmp var, 'A'

Compara o conteúdo da posição de memória var com o caracter ASCII 'A'

Instruções usadas no controle de fluxo (2)

- Registrador de Flags: Consiste em um grupo individual de bits de controle (flag) [O D I T S Z A P C]



OF (Overflow Flag): Setada quando ocorre overflow aritmético.

DF (Direction Flag): Setada para auto-incremento em instruções de string.

IF (Interruption Flag): Permite que ocorram interrupções quando setada.Pode ser setada pelo sistema ou pelo usuário.

TF (Trap Flag) (debug): Usada por debugadores para executar programas passo a passo.

SF (Signal Flag): Resetada (SF=0) quando um resultado for um número positivo ou zero e setada (SF=1) quando um resultado for negativo.

ZF (Zero Flag): Setada quando um resultado for igual a zero.

AF (Auxiliar Flag): Setada quando há "vai um" na metade inferior de um byte.

PF (Parity Flag): Setada quando o número de bits 1 de um resultado for par.

CF (Carry Flag): Setada se houver "vai um" no bit de maior ordem do resultado. Também usada por instruções para tomadas de decisões.

Instruções usadas no controle de fluxo (3)

- **`CMP destino, origem`**

Esta instrução subtrai o operador origem do destino (destino – origem), mas não armazena o resultado da operação, apenas afeta o estado das flags de estado

Flags setadas de acordo com o resultado de: destino - origem

Se destino-origem == 0 (destino==origem)
ZF=1, SF=0, CF=0
Se destino-origem < 0 (destino<origem)
ZF=0, SF= 1, CF=1
Se destino-origem > 0 (destino>origem)
ZF=0, SF= 0, CF=0

Instruções usadas no controle de fluxo (4)

- **`JXXX  rótulo_de_destino`**

  Instrução de Saltos condicional, XXX é uma condição dependente de algum dos Flags de Estado

  Se a condição XXX é verdadeira:

  - a próxima instrução a ser executada é aquela definida pelo rótulo_de_destino;
  - a CPU ajusta o registrador IP para apontar para a posição de memória dada por rótulo_de_destino.

  Se a condição XXX é falsa:

  - a próxima instrução é aquela que imediatamente segue o salto.

Instruções usadas no controle de fluxo (5)

- **JE rótulo** (Jump if Equal ... JZ)
  ZF = 1 --> Salta se A == B
- **JNE rótulo** (Jump if not equal...JNZ)
  ZF = 0 --> **Salta se A != B**
- **JA rótulo** (Jump if Above)
  (CF=0) AND (ZF=0) --> Salta se A>B
- **JAE rótulo** (Jump if Above or Equal)
  CF=0 --> Salta se A>=B
- **JB rótulo** (Jump if Below)
  CF=1 --> Salta se A<B
- **JBE rótulo** (Jump if Below or Equal)
  (CF=1) OR (ZF=1) --> Salta se A<=B

CMP A , B

Exemplo 5

- Supondo que AX e BX contenham números inteiros sem sinais, escreva um trecho de programa que coloque o maior deles em CX.

```
        ...
        MOV   CX,AX      ;AX já é pressuposto ser o maior deles
        CMP   AX,BX
        JAE   ABAIXO     ;Salta se AX >= BX
        MOV   CX,BX      ;caso BX seja de fato o maior deles
ABAIXO: ...              ;continuação do programa
        ...
```

Instruções usadas no controle de fluxo (6)

- O comando "if" de uma linguagem de alto nível como C, em Assembly é a junção do **CMP** com um **"jump" condicional**.
- Exemplo: Em linguagem de alto nível:

```
IF      AL (menor ou igual a)  BL
        THEN    (exibir AL)
        ELSE    (exibir BL)
END_IF
```

```
        .....
        CMP AL,BL       ;if AL menor ou igual a BL
        JA  TROCA       ;//jump if AL>BL

        MOV DL,AL       ;then
        INT 21h
        JMP FIM

TROCA:  MOV DL,BL       ;else
        INT 21h

FIM:    .....           ;end_if
```

Instruções para Laços (1)

- **`LOOP símbolo`**

  A instrução LOOP decrementa CX de 1 e transfere a execução do programa para o símbolo que é dado como operador, caso CX ainda não seja 1.

  Cx ← Cx – 1
  Se (Cx != 0)
  Jmp *rótulo*

  Observação:

  Se Cx = 0 no início serão feitos 65.536 loops !

Exemplo 6

- Faça um programa em ASM86 correspondente ao seguinte "código C"

```
unsigned int count = 5;
unsigned int x=10;
main()
{
   do{
        x++;
        count--;
   }while (count>0)

}
```

```
.MODEL SMALL
.STACK
.CODE
        MOV CX,count    ; CX <- count
L1:     INC x           ; x++
        LOOP  L1        ; repete a partir de L1

.DATA
x       dw      10
count   dw      5

END
```

Instruções para Laços (2)

```
unsigned int count = 5;
unsigned int x=10;
main()
{
    do{
        x++;
        count--;
    }while (count>0)

}
```

- Também pode-se usar **CMP e JXXX**

  No exemplo anterior:

```
.MODEL SMALL
.STACK
.CODE

L1:     INC x           ; x++
        DEC count       ; count--
        CMP count,0
        JA  L1          ; salta se count>0

.DATA
x       db      10
count   db      5

END
```

Exemplo 7

- Faça um programa em assembly equivalente ao seguinte programa em C:

```
int y=0, i, n = 7;
main( )
{
for ( i = 1; i <= n; i ++)
y = y + i;
}
```

```
.MODEL SMALL
.STACK
.CODE
        MOV BX,y        ; BX <- y
        MOV i,1         ; i=1
        MOV DX,i        ; DX <- i
FOR:    CMP DX,n        ; compara DX(ou i) e n
        JA END_FOR      ; salta se DX > n
        ADD BX,DX       ; BX = BX + DX
        INC DX          ; DX (ou i) + 1
        JMP FOR         ; salta p/ FOR
        MOV y,BX        ; coloca o valor final de BX em y
        MOV i,DX        ; coloca o valor final de DX em i
.DATA
y       dw      0
i       dw      ?
n       dw      7
END
```

Interrupções de Software (1)

- Interrupções de software podem ser ativadas diretamente por nossos programas assembly
- Dois tipos de interrupções
  - Interrupções do Sistema Operacional DOS
  - Interrupções da BIOS
- Para gerar interrupções do DOS use: INT 21h
- Quando usamos esta instrução, o DOS chama uma rotina de tratamento específica, dependendo do tipo de interrupção
- O tipo de interrupção será definido em função do valor que estiver armazenado no registrador AL

Interrupções de Software (2)

- INT 21h (Entrada: AH <- 01h , Saída: caracter ->AL)
  - Lê um caracter da console e coloca o seu código ASCII no registrador AL
  - Como gerar este tipo de Interrupção
    1. Copie o valor 08h dentro do registrador AH
    2. Chame a instrução "INT 21h"

       Após esta chamada, assim que for digitado um caracter, seu código ASCII será colocado dentro de AL

- INT 21h (Entradas: AH <- 02h , DL<- caracter)
  - Imprime o caracter ou o executa, se for do tipo beep, line feed ou assemelhados
  - Como gerar este tipo de Interrupção
    1. Copie o valor 02h dentro do registrador AH
    2. Copie o código ASCII do caracter que deseja imprimir dentro do registrador DL
    3. Chame a instrução "INT 21h"

Exemplo 8

- Escreva um trecho de programa em assembly que leia dois caracteres e os imprime na ordem inversa em que foram lidos

```
...
                ; --- Leitura dos 2 caracteres ---
MOV AH,01h      ; Função 1 do DOS (leitura de caractere)
INT 21h         ; lê 1o caracter, retorna código ASCII ao registrador AL
MOV BL,AL       ; move o código ASCII para o registrador BL por enquanto
INT 21h         ; lê 2o caracter, retorna código ASCII ao registrador AL

                ; --- Impressão dos 2 caracteres, na ordem invertida ---
MOV AH,02h      ; Função 2 do DOS (escrita de caractere)
MOV DL,AL       ; move o código ASCII do 2o caractere lido p/ DL
INT 21h         ; imprime o caractere cujo codigo está em DL
MOV DL,BL       ; move o código ASCII do 1o caracter lido p/ DL
MOV AH,2h       ; função 2h, imprime caracter
INT 21h         ; imprime o caractere cujo codigo está em DL
...
```

Usando Procedimentos

- Declarando um procedimento
  - Na declaração, a primeira palavra, NomePr, corresponde ao nome do procedimento
  - A diretiva Ret carrega IP com o endereço armazenado na pilha para retornar ao programa que o chamou
  - NomePr EndP indica o fim do procedimento.

```
NomePr  Proc            ; Declaração do Procedimento “Intrasegment”

        ...             ; Conteúdo do Procedimento...

        Ret             ; Diretiva de retorno
NomePr  EndP            ; Fim do Procedimento
```

- Usando um procedimento

```
        CALL NomePr     ; Chamando o procedimento
```

Exemplo 9

- Escreva um programa contendo uma rotina que soma dois bytes armazenados em AH e AL, e o resultado da soma em BX. O programa deve colocar os valores 5 e 10 nos registradores AH e AL, chamando em seguida a função. No final jogue o resultado da soma na memória.

```
.MODEL SMALL
.STACK
.CODE
        MOV AX,050Ah    ; ???
        CALL Soma
        MOV result,BL   ;copia resultado no endereço de memória BX
Soma    Proc            ; Declaração do Procedimento
        Mov BX, 0       ; Conteúdo do Procedimento...
        Mov BL, AH
        Add BL, AL
        Ret             ; Diretiva de retorno
Soma    EndP            ; Fim do Procedimento
.DATA
result  DB      ?
```

Manipulação de pilha

PUSH

⇒Empilha uma word

Sintaxe: **Push** *origem*

Lógica:

SP ← SP – 2

(SP) ← *origem*

Operações	Exemplo
Reg	PUSH BX
Memória	PUSH Var
Reg_Seg (exceto IP)	PUSH ES

POP

⇒Desempilha uma word

Sintaxe: **Pop** *destino*

Lógica:

destino ← (SP)

SP ← SP + 2

Operações	Exemplo
Reg	POP SI
Memória	POP Var
Reg_Seg (exceto CS e IP)	POP DS

Exemplo 10

- Usando a pilha para armazenar valores temporariamente antes chamar algum procedimento

```
...
        XOR DX,DX
        MOV AX,10
        PUSH AX
        CALL Metade
        MOV result,AX  ;copia resultado no endereço de memória
        POP AX         ; restaura o valor de AX p/ o de antes
...
Metade  PROC           ; Declaração do Procedimento
        DIV 2          ;
        RET            ; Diretiva de retorno
Metade  EndP           ; Fim do Procedimento
...
```