MC404

ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

2006

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC404

ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

"Pilha, Sub-Rotina, *Arrays* e Modos de Endereçamento"

Pilha, Sub-Rotina, Arrays e Modos de Endereçamento
Sumário

- Pilha
  - PUSH e POP
- Sub-Rotina
  - Call e Ret
- Arrays
- Modos de Endereçamento

PILHA

- Organização da Pilha (*stack*)
  - estrutura de dados de uma dimensão organizada em algum trecho (segmento) da Memória;
  - o primeiro item adicionado é o último a ser removido (*first-in, last-out*);
  - a posição da pilha mais recentemente acrescida é o topo da pilha.

PILHA

- Declaração do segmento de pilha

```
.STACK  100h          ;dimensiona o tamanho da pilha
```

  - SS -> aponta o início do segmento de pilha (base)
  - SP -> aponta o topo da pilha (define o deslocamento do topo em relação à base)

- A pilha cresce do topo para baixo.
  - Endereço para acesso à pilha: SS:SP (no. de segmento:offset)

PILHA
Instruções para manipular dados na pilha

PUSH fonte

PUSHF

- Onde fonte é:
  - um registrador de 16 bits;
  - uma palavra de memória ou variável de 16 bits (de tipo DW).

- A execução de PUSH resulta nas seguintes ações:

  - o registrador SP (*stack pointer*) é decrementado de 2;
  - uma cópia do conteúdo da fonte é armazenado na pilha de forma que:

    - posição SS:SP → armazena o byte baixo da fonte;
    - a posição SS:(SP + 1) → armazena o byte alto;
    - o conteúdo da fonte não é alterado.

PILHA
Instruções para manipular dados na pilha

- A execução de PUSHF , que não possui operando, resulta:
  - o registrador SP (*stack pointer*) é decrementado de 2
  - uma cópia do conteúdo do registrador de FLAGS é armazenado na pilha
- Observações:
  - As instruções de pilha não alteram os FLAGS;
  - Não é possível movimentar dados de 8 bits, nem valores imediatos.

PILHA
Instruções para manipular dados na pilha

Exemplo de operação: ...

```
PUSH   AX    ;instrução 1
PUSHF        ;instrução 2
```

Offset	Antes	Depois de 1	Depois de 2	
0100h	? ← SP	?	?	
00FFh	?	12h	12h	AX
00FEh	?	34h ← SP	34h	1234h
00FDh	?	?	56h	
00FCh	?	?	78h ←SP	
00FBh	?	?	?	FLAGS
00FAh	?	?	?	5678h
00F9h	?	?	?	
...				
(Base)				

PILHA
Instruções para manipular dados na pilha

POP destino

POPF

- **Onde destino é:**
  - um registrador de 16 bits;
  - uma palavra de memória ou variável de 16 bits (de tipo DW).

- **A execução de POP resulta nas seguintes ações:**
  - o conteúdo das posições SS:SP (byte baixo) e SS:(SP + 1) (byte alto) é movido para o destino;
  - o registrador SP (*stack pointer*) é incrementado de 2.

PILHA
Instruções para manipular dados na pilha

- **A execução de POPF , que não possui operando, resulta:**
  - **o conteúdo das posições SS:SP (byte baixo) e SS:(SP + 1) (byte alto) é movido para o registrador de FLAGS;**
  - **o registrador SP (*stack pointer*) é decrementado de 2.**

PILHA

Instruções para manipular dados na pilha

Exemplo de operação:

```
POPF          ;instrução 1
POP   AX      ;instrução 2
```

Offset	Antes	Depois de 1	Depois de 2
0100h	?	?	? ← SP
00FFh	12h	12h	12h
00FEh	34h	34h ← SP	34h
00FDh	56h	56h	56h
00FCh	78h ← SP	78h	78h
00FBh	?	?	?
00FAh	?	?	?
00F9h	?	?	?
.....			
(Base)			

AX antes F0D3h

depois 1234h

FLAGS antes 006Ah

Depois 5678h

Um exemplo de uso de pilha:

```
TITLE ENTRADA INVERTIDA
.MODEL  SMALL
.STACK  100h
.CODE
        MOV AH,2        ;exibe o Prompt para o usuario
        MOV DL,'?'      ;caracter '?' para a tela
        INT 21h         ;exibe
        XOR CX,CX       ;inicializando contador caracteres em zero
        MOV AH,1        ;prepara para ler um caracter do teclado
        INT 21h         ;caracter em AL
;while   caracter nao e' <CR>   do
INICIO: CMP AL,0DH      ;e' o caracter <CR>?
          JE PT1        ;sim, entao saindo do loop
;salvando o caracter na pilha e incrementando o contador
          PUSH AX       ;AX vai para a pilha (interessa somente AL)
          INC CX        ;contador = contador + 1
;lendo um novo caracter
          INT 21h       ;novo caracter em AL
          JMP INICIO    ;retorna para o inicio do loop
;end_while
```

Um exemplo de uso de pilha:

```
PT1:  MOV AH,2          ;prepara para exibir
      MOV DL,0DH        ;<CR>
      INT 21h           ;exibindo
      MOV DL,0AH        ;<LF>
      INT 21h           ;exibindo: mudança linha
      JCXZ FIM          ;saindo se nenhum caracter foi digitado

;for   contador vezes   do
TOPO:  POP DX           ;retira primeiro caracter  da pilha
       INT 21h          ;exibindo caracter
       LOOP TOPO        ;em loop até CX = 0
;end_for

FIM:   MOV AH,4CH       ; saída para o DOS
       INT 21H
       END
```

Sub-Rotinas (ou *procedures*)

- **Sintaxe para subrotinas:**

```
nome    PROC  tipo
;
;corpo da subrotina - instruções
;
        RET   ;transfere o controle de volta para a
rotina principal
nome    ENDP
```

- **Tipos possíveis:**
  - NEAR : subrotina no mesmo segmento de código
  - FAR : em outro segmento de código

Sub-Rotinas (ou *procedures*)

- **Mecanismo de chamada e retorno:**

```
PRINCIPAL   PROC
...
   CALL   SUB1
;próxima instrução
...
PRINCIPAL  ENDP

SUB1              PROC
;primeira instrução
...
   RET
SUB1              ENDP
```

Sub-Rotinas (ou *procedures*)

- **Comunicação de dados entre subrotinas:**
  - **Em Linguagem Montadora, não há lista de parâmetros;**
  - **Se há poucos valores de entrada e saída → usar registrador**
  - **Se há muitos valores de entrada e saída → usar a pilha**

Sub-Rotinas (ou *procedures*)

Chamada e retorno de subrotinas

- **Instrução de chamada:** `CALL nome`
  - IP, que contem o *offset* do endereço da próxima instrução da rotina "chamante" (após a instrução CALL), é armazenado na pilha;
  - IP recebe o *offset* do endereço da primeira instrução da subrotina chamada.

- **Instrução de retorno:** `RET`
  - Faz com que o *offset* do endereço da próxima instrução da rotina "chamante", que está na pilha, seja recarregado em IP.

- Ambos CALL e RET não afetam FLAGS.

Sub-Rotinas (ou *procedures*)

- **Mecanismo de chamada:**

Offset	Seg. de Código	Antes	Depois		
	MAIN PROC				
	...		IP	IP	
	...		1012h	1200h	
	CALL SUB1				
1012h	;próxima instrução		Pilha	Pilha	
		SP →	?	?	0100h
			?	12h	00FFh
	SUB1 PROC		?	SP → 10h	00FEh
1200h	;primeira instrução		?	?	00FDh
	...		?	?	00FCh
	...		?	?	00FBh
1300h	RET		?	?	00FAh

Sub-Rotinas (ou *procedures*)

Mecanismo de retorno:

Offset	Seg. de Código	Antes	Depois	
	MAIN PROC			
	...	IP	IP	
	...	1300h	1012h	
	CALL SUB			
11012h	;próxima instrução	Pilha	Pilha	
		?	SP → ?	0100h
		12h	12h	00FFh
	SUB1 PROC	SP → 10h	10h	00FEh
1200h	;primeira instrução	?	?	00FDh
	...	?	?	00FCh
	...	?	?	00FBh
1300h	RET	?	?	00FAh

Exemplo

```
TITLE   MULTIPLICACAO POR SOMA E DESLOCAMENTO
.MODEL  SMALL
.STACK  100h
.CODE
PRINCIPAL    PROC
     ...       ;supondo a entrada de dados
       CALL  MULTIPLICA
     ...       ;supondo a exibição do resultado
       MOV AH,4Ch
       INT 21h
PRINCIPAL           ENDP

MULTIPLICA          PROC
;multiplica dois numeros
;A e B por soma e deslocamento
;entradas:              AX = A, BX = B,
; numeros na faixa 00h - FFh
;saida:      DX = A*B (produto)
              PUSH AX
              PUSH BX    ;salva os conteudos
                         ;de AX e BX
              AND DX,0    ;inicializa DX em 0
```

Exemplo (cont.)

```
;repeat
;if   B e' impar
TOPO:   TEST BX,1 ;B e' impar?
        JZ PT1      ;nao, B e' par (LSB = 0)
;then
        ADD DX,AX   ;sim, entao
                    ;produto = produto + A
;end_if
PT1:    SHL AX,1        ;desloca A para
                    ; a esquerda 1 bit
        SHR BX,1        ;desloca B para a
                    ;direita 1 bit
;until
        JNZ TOPO        ;fecha o loop repeat
        POP BX
        POP AX          ;restaura os
                    ;conteudos de BX e AX
        RET             ;retorno para o
                    ;ponto de chamada
MULTIPLICA      ENDP
END  PRINCIPAL
```

Arrays

- *Arrays* unidimensionais
  - *Array* uni-dimensional: é uma lista ordenada de elementos do mesmo tipo.
  - Por ordem entende-se que há um primeiro, um segundo, um terceiro elemento, e assim por diante até um último elemento;
  - Em notação matemática, um *array* A de seis elementos é denotado por:

    A[1], A[2], A[3], A[4], A[5], A[6]

Arrays

Exemplos de *arrays*:

```
MSG   DB    'abcde'      ; array composto por um string de 5 caracteres
                         ; ASCII
W     DW    1010h,1020h,1030h    ;array de 3 valores de 16 bits
```

Onde W se situa na memória a partir do *offset* de endereço, por exemplo 0200h como:

	Offset de endereço	Endereço simbólico	Conteúdo
W →	0200h	W	10h
	0201h		10h
	0202h	W+2	20h
	0203h		10h
	0204h	W+4	30h
	0205h		10h

Arrays

Operador DUP: define *arrays* com valores repetidos (duplicatas)

```
GAMA  DB    100 DUP (0)   ;cria um array de 100 bytes
                          ;com valor inicial zero, a partir
                          ;do offset definido por GAMA
BETA  DW    200 DUP (?)   ;cria um array de 200 words (16
                          ;bits) não inicializados, a partir
                          ;do offset  BETA

LINHA DB    5, 4, 3 DUP (2, 3 DUP (0), 1)  ;DUP's encadeados
```

Equivalente a definição de LINHA dada abaixo:

```
LINHA DB    5, 4, 2, 0, 0, 0, 1, 2, 0, 0, 0, 1, 2, 0, 0, 0, 1
```

Modos de Endereçamento

A forma como um operando é especificado numa instrução é conhecido como Modo de Endereçamento.

Modos de enderçamento:

1. Modo Registrador: o operando é um registrador da CPU;
2. Modo Imediato: o operando é uma constante fornecida imediatamente;
3. Modo Direto: o operando é uma variável declarada no .DATA, ou seja uma posição de memória com endereço bem determinado;
4. Modo Registrador Indireto;
5. Modo Base;
6. Modo Indexado;
7. Modo Base Indexado;

Modos de Endereçamento

1. Modo Registrador: o operando é um registrador da CPU.

   Exemplo: ADD AX,BX

2. Modo Imediato: o operando é uma constante fornecida imediatamente.

   Exemplo: MOV AX,5

Modos de Endereçamento

3. Modo Direto: o operando é uma variável declarada no .DATA, ou seja uma posição de memória com endereço bem determinado.

```
DADO    DB     5
......
MOV AL, [DADO]
```

Modos de Endereçamento

4. Modo Registrador Indireto: O *offset* do endereço do operando está contido num registrador.

O registrador especificado atua como ponteiro para a posição de memória

Formato do operando: [registrador]

Registradores utilizados:

BX, SI, DI juntamente com o registrador de segmento DS o endereço é formado por:

DS:[registrador]

BP juntamente com o registrador de segmento SS o endereço é formado por:

SS:[BP]

Modos de Endereçamento

Exemplo1: Supondo que SI = 0100h e que a palavra contida na posição de memória de *offset* 0100h seja 1234h:

```
MOV AX,SI          ;AX recebe 0100h
MOV AX,[SI]        ;AX recebe 1234h
```

Modos de Endereçamento

Exemplo 2: Escreva um trecho de programa que acumule em AX a soma dos 10 elementos do *array* LISTA abaixo:

```
        LISTA DW 10,20,30,40,50,60,70,80,90,100
        ...

        XOR AX,AX       ; inicializa AX com zero
        LEA SI,LISTA    ; SI recebe o offset de end. de LISTA
        MOV CX, 10      ; contador inicializado no. de elementos
SOMA:   ADD AX,[SI]     ; acumula AX com o elemento de LISTA
                        ; apontado por SI
        ADD SI,2        ; movimenta o ponteiro para o próximo
        LOOP  SOMA      ; faz o laço até CX = 0
              ...
```

Modos de Endereçamento

5. Modo Base (*Based Mode*)

O *offset* do endereço do operando é obtido adicionando um deslocamento ao conteúdo de um registrador base

O deslocamento pode ser:

- O *offset* do endereço de uma variável;
- Uma constante (positiva ou negativa);
- O *offset* do endereço de uma variável mais ou menos uma constante;

Formatos possíveis do operando:

[registrador + deslocamento] ou [deslocamento + registrador]

[registrador] + deslocamento ou deslocamento + [registrador]

deslocamento [registrador]

Modos de Endereçamento

Registradores utilizados:

BX (*base register*) juntamente com o registrador de segmento DS

BP (*base pointer*) juntamente com o registrador de segmento SS

Exemplo1: Supondo que BX contém o valor 4d:

```
LISTA     DW    10,20,30,40,50,60,70,80,90,100

          MOV AX, [LISTA + BX]    ; resulta que AX = 0030
          ...
```

Modos de Endereçamento

Exemplo 2: Escreva um trecho de programa que acumule em AX a soma dos 10 elementos do *array* LISTA abaixo, usando endereçamento Base:

```
LISTA       DW      10,20,30,40,50,60,70,80,90,100
    ...
    XOR AX,AX       ; inicializa AX com zero
    XOR BX,BX       ; limpa o registrador base
    MOV CX, 10      ; contador inicializado no. de elementos

SOMA: ADD AX,LISTA+[BX]   ; soma AX com o elemento de LISTA
                          ; apontado por offset de LISTA + BX
    ADD BX,2              ; incrementa base
    LOOP  SOMA            ; faz o laço até CX = 0
```

Modos de Endereçamento

6. Modo Indexado (*Indexed Mode*)

O *offset* do endereço do operando é obtido adicionando um deslocamento ao conteúdo de um registrador indexador

As opções de deslocamento são as mesmas do Modo Base

Formatos possíveis do operando:

[registrador + deslocamento] ou [deslocamento + registrador]
[registrador] + deslocamento ou deslocamento + [registrador]
deslocamento [registrador]

Registradores utilizados:

SI e DI juntamente com o registrador de segmento DS

Modos de Endereçamento

Exemplo1: Supondo que SI contenha o *offset* de endereço de LISTA:

```
LISTA DW    10,20,30,40,50,60,70,80,90,100

    LEA SI,LISTA          ; SI recebe o offset de LISTA
    MOV AX, [SI + 12]     ; resulta que AX = 70
```

Modos de Endereçamento

Exemplo 2: Converta a mensagem abaixo para maiúsculas:

```
MSG   DB     'isto e uma mensagem'
      ...
      MOV CX,19    ; inicializa no. de carateres de MSG
      XOR SI,SI    ; SI = 0

TOPO:   CMP MSG[SI], ' '   ; compara caracter com branco
        JE  PULA           ; igual, não converte
        AND MSG[SI],0DFh ; diferente, converte para maiúscula

PULA:   INC SI             ; incrementa indexador
        LOOP TOPO          ; faz o laço até CX = 0
             ...
```

Modos de Endereçamento

7. Modo por Base Indexado (*Based Indexed Mode*)

- *O offset* do endereço do operando é obtido somando:
  - o conteúdo de um registrador de base (BX ou BP);
  - o conteúdo de um registrador índice (SI ou DI);
  - opcionalmente, o *offset* de endereço de uma variável;
  - opcionalmente, uma constante (positiva ou negativa).
- Formatos possíveis do operando:
  - variável [reg_de_base] [reg_índice]
  - variável [reg_de_base + reg_índice + constante]
  - constante [reg_de_base + reg_índice + variável]
  - [reg_de_base + reg_índice + variável + constante]

Modos de Endereçamento

Registradores utilizados:

SI, DI e BX juntamente com o registrador de segmento DS
SI, DI e BP juntamente com o registrador de segmento SS

Exemplo1: Supondo a variável LISTA abaixo, e que SI = 14 e BX = 2:

```
LISTA      DW    10,20,30,40,50,60,70,80,90,100
MOV AX, LISTA [BX][SI] ; resulta que AX = LISTA+2+14 =
                       ; 90
```

Modos de Endereçamento

Exemplo 2: Suponha que A é uma matriz 3X4 com elementos de tipo DW. Escreva um trecho de programa que zere os elementos da 2a. linha de A.

```
...
        MOV BX,8    ; BX indica o 1o. elemento da linha 2
        MOV CX,4    ; CX contem o número de elementos de linha
        XOR SI,SI   ; inicializa o indexador de coluna
LIMPA: MOV A [BX][SI], 0  ; carrega zero no operando
                          ; calculado
       ADD SI,2     ; incrementa 2 em SI -> tipo DW = 2 bytes
       LOOP LIMPA   ; faz o laço até que CX seja zero
...
```

Modos de Endereçamento

Observação: Acessando a pilha por meio de endereçamento por base:

SP sempre aponta para o topo da pilha

Problema: como obter cópias de dados existentes na pilha sem modificá-la

Solução: endereçamento por base usando BP (*base pointer*)

Modos de Endereçamento

Exemplo: Escreva um trecho de programa que carregue AX, BX e CX com as três palavras mais superiores da pilha, sem modificá-la.

```
MOV BP,SP        ; BP aponta para o topo da pilha
MOV AX, [BP]     ; coloca o conteúdo do topo em AX
MOV BX, [BP+2]   ; coloca a 2a. palavra (abaixo do topo)
                 ; em BX
MOV CX, [BP+4]   ; coloca o 3a. palavra em CX
```

OBS.: Usado por compiladores para acesso a parâmetros passados na pilha em funções/procedimentos

Modos de Endereçamento

Neste caso, no Modo de endereçamento por Base:

- BP pode especificar o offset de um endereço na pilha;
- Pode ser somado um deslocamento (positivo ou negativo;
- O operando final especificado pode navegar para dentro da pilha;

Segment Override

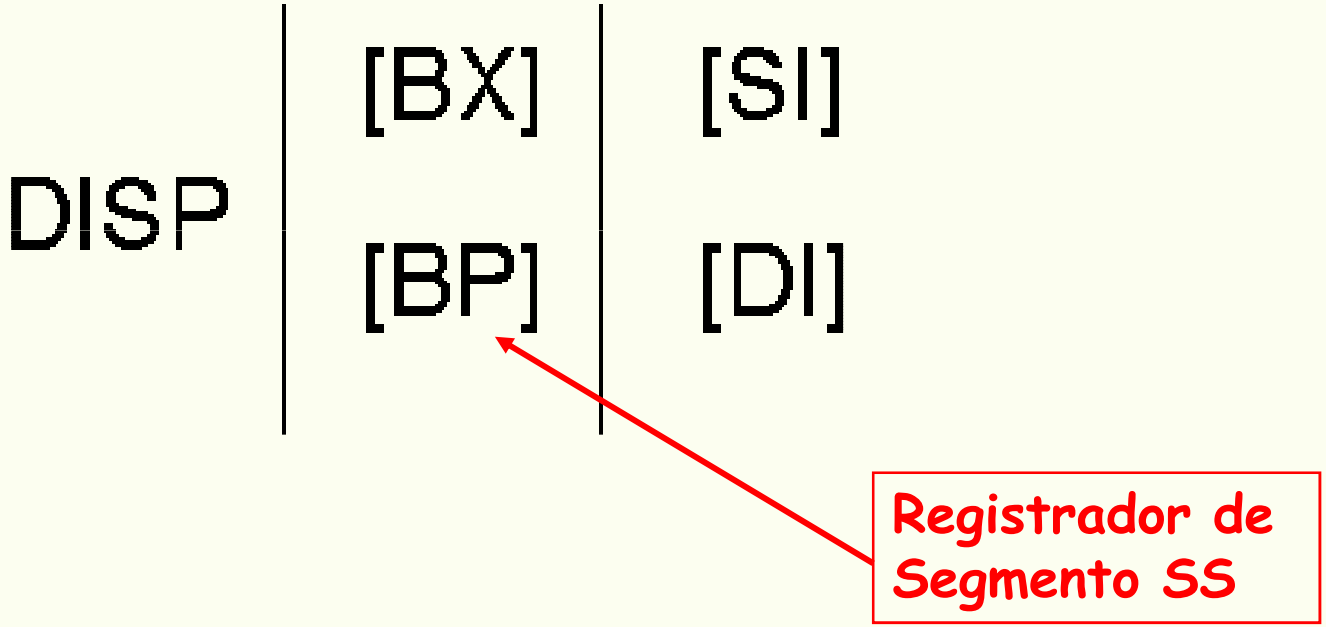
Se for necessário acessar dados em outro segmento diferente do apontado por DS, por exemplo ES:

```
MOV AX, ES:[SI]
```

Pode-se utilizar *segment override* nos modos de registro indireto, por base e indexado.

Modos de Endereçamento
Resumo

- Formato do Endereçamento no 8086



- Registrador de Segmento padrão DS
- Pode ser usado ES com o mecanismo de *Segment Override*

Modos de Endereçamento
Diretivas

PTR

- A instrução MOV [BX],1 é ilegal, pois o Montador não pode determinar se [BX] aponta para uma informação na memória do tipo byte ou do tipo word.

Soluções:

```
MOV   BYTE  PTR [BX],1  ;define o destino como byte

MOV   WORD PTR [BX],1 ;define o destino como word
```

Modos de Endereçamento
Diretivas

LABEL

É uma pseudo-instrução para alterar tipo de variáveis (*override*)

Exemplo:

```
TEMPO       LABEL WORD
HORAS       DB    10
MINUTOS     DB    20
```

Esta declaração feita no segmento de dados (.DATA), permite que:

TEMPO e HORAS recebam o mesmo endereço pelo Montador;

TEMPO (16 bits) engloba HORAS e MINUTOS (8 + 8 bits);

são legais as seguintes instruções:

```
MOV   AH,HORAS
MOV   AL,MINUTOS
```

e

```
MOV   AX,TEMPO      ;produz o mesmo efeito das acima
```