

7. A pilha e subrotinas

7.1 Organização da Pilha (*stack*)

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**.

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)
- Movimentar dados para pilha: PUSH fonte,
PUSHF
- Movimentar dados da pilha: POP destino,
POPF
- As instruções de pilha **não alteram** os FLAGS
- Não é possível movimentar dados de 8 bits, nem valores imediatos

Instruções para colocar 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
 - a 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

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

Exemplo de operação:

```

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

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

Instruções para retirar 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

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

Exemplo de operação:

```

...
POPFB             ;instrução 1
POP AX           ;instrução 2
    
```

Offset	Antes		Depois de 1		Depois de 2		AX antes
0100h	?		?		?		F0D3h
00FFh	12h		12h		12h		depois
00FEh	34h		34h	<- SP	34h		1234h
00FDh	56h		56h		56h		
00FCh	78h	<- SP	78h		78h		FLAGS
00FBh	?		?		?		antes
00FAh	?		?		?		006Ah
00F9h	?		?		?		depois
...							5678h
(Base)							

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 de 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
PT1:    MOV AH,2      ;prepara para exibir
        MOV DL,0DH    ;<CR>
        INT 21h       ;exibindo
        MOV DL,0AH    ;<LF>
        INT 21h       ;exibindo: mudança de linha
        JCXZ FIM      ;saindo se nenhum caracter foi digitado
;for  contador vezes  do
TOPO:   POP DX        ;retira o primeiro caracter da pilha
        INT 21h       ;exibindo este caracter
        LOOP TOPO     ;em loop até CX = 0
;end_for
FIM:    MOV AH,4CH    ;preparando para sair para o DOS
        INT 21H
        END

```

7.2 Terminologia para subrotinas (ou *procedures*)

Sintaxe para subrotinas:

```

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

```

Obs: tipos possíveis **NEAR** -> subrotina no mesmo segmento de código
FAR -> em outro segmento de código

Mecanismo de chamada e retorno:

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

SUB1 PROC
;primeira instrução
...
RET
SUB1 ENDP

Comunicação de dados entre subrotinas:

- em Linguagem Montadora, **não há** lista de parâmetros;
- se há poucos valores de entrada e saída -> **usar registradores**

7.3 Chamada e retorno de subrotinas

Instrução de chamada:

CALL nome

- **IP**, que contém 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.

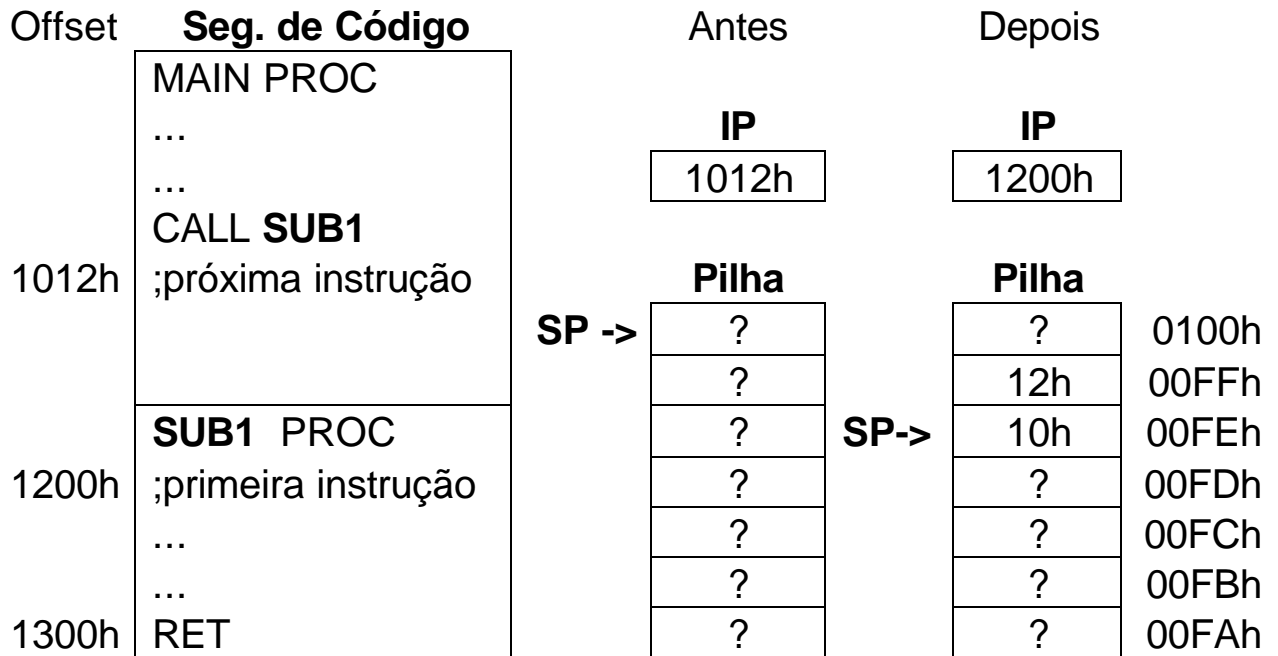
Instrução de retorno:

RET

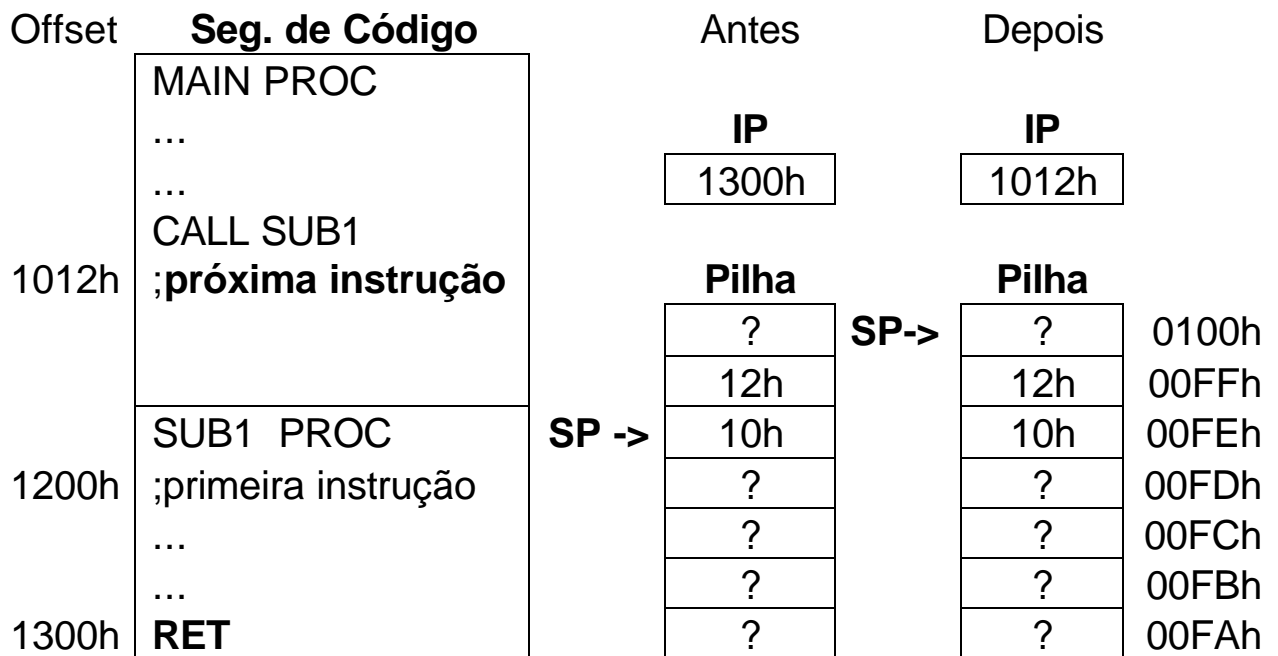
- faz com que o **offset do endereço** da próxima instrução da rotina "chamante", que está na **pilha**, seja recarregado em **IP**.

Ambos **CALL** e **RET** não afetam FLAGS.

Mecanismo de chamada:



Mecanismo de retorno:



Um exemplo de subrotina:

```

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
;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

```


Exercícios sugeridos:

1) Suponha que AX = 1234h, BX = 5679h, CX = 9ABCh e SP = 0100h. Dê o conteúdo de AX, BX, CX e SP após a execução do seguinte trecho de programa:

```
...  
PUSH AX  
PUSH BX  
XCHG AX,CX  
POP CX  
PUSH AX  
POP BX
```

2) Escreva algumas linhas de programa para:

a) colocar o conteúdo do topo da pilha em AX, sem modificar o conteúdo e a posição do topo da pilha;

b) colocar em CX a palavra que esteja abaixo daquela do topo da pilha (você poderá usar AX como registrador auxiliar;

c) troque os conteúdos das duas palavras do topo da pilha (a do topo e a logo abaixo desta - você poderá usar AX e BX como auxiliares).

3) O seguinte método pode ser usado para gerar números aleatórios na faixa de 1 a 32767:

- inicie com qualquer número na faixa acima;
- desloque uma casa binária à esquerda;
- substitua o bit 0 pelo XOR dos bits 14 e 15;
- limpe o bit 15.

Escreva as seguintes rotinas (*procedures*):

a) subrotina LEIA que permita ao usuário entrar um número binário e armazená-lo em AX (baseie-se no trecho para entrada binária da página 6.16);

- b) subrotina ALEATORIO que recebe um número em AX e retorna um número aleatório também em AX, segundo o método acima apresentado;
- c) subrotina ESCREVA que exibe AX em binário no monitor (baseie-se no trecho de saída binária da página 6.17);
- d) escreva uma rotina PRINCIPAL que emita uma mensagem ao usuário para entrar com o número binário inicial de 16 bits, chame a subrotina LEIA para ler tal número, chame as subrotinas ALEATORIO e ESCREVA para calcular e exibir 100 números aleatórios, que devem ser apresentados na tela cada um com 16 bits seguidos, 4 por linha, cada grupo separado por 4 espaços em branco.