## Operacje na stosie

Stos jest obszarem pamięci o dostępie LIFO (Last Input – First Output). Adresowany jest niejawnie przez rejestr segmentowy **SS** oraz wskaźnik wierzchołka stosu **SP**. Używany jest do przechowywania stanu procesora podczas wykonywania procedury i obsługi przerwań oraz do przekazywania parametrów do procedury.

**PUSH** ***src*** - instrukcja składowania na stosie działa następująco

- odejmuje **2** od zawartości rejestru **SP**
- zapisuje zawartość ***src*** na stosie pod adresem **SS:SP**.

**POP** ***dst*** - instrukcja pobierania ze stosu działa następująco

- pobiera ze stosu wartość spod adresu **SS:SP** i zapisuje do ***dst***.
- dodaje **2** do rejestru **SP**.

**src** oraz **dst** mogą określać rejestr 16-bitowy lub komórkę pamięci (na stosie umieszczane są wartości 16-bitowe).
Dla programów typu program.com rejestr segmentowy stosu **SS** ustawiany jest na taką samą wartość jak rejestr segmentu kodu **CS** zaś zawartość **SP** przyjmuje wartość **0FFFEh**. Pod adresem **SS: 0FFFEh** przechowywany jest adres powrotu z programu po instrukcji **ret**.
Można obserwować zawartość stosu oraz zachowanie wskaźnika stosu naciskając przycisk [**Stack**] w oknie emulatora. Pozycja wierzchołka stosu wskazywana jest przez symbol <.

## Zadanie

Zaobserwuj zachowanie procesora i stosu podczas wykonywania następujących programów

```
ORG   100h
   MOV   AX, 1234h
   PUSH  AX        ; zapisanie zawartości  AX na stosie
   MOV   AX, 5678h  ; zmiana zawartości AX
   POP   AX        ; odtworzenie oryginalnej zawartości  AX
RET
END
```

```
ORG   100h
   MOV   AX, 1234h  ; zapisanie wartości 1234h do AX.
   MOV   BX, 5678h  ; zapisanie wartości 5678h do BX
   PUSH  AX        ; zapisanie zawartości AX na stosie
   PUSH  BX        ; zapisanie zawartości BX na stosie
   POP   AX        ; zapisanie do AX oryginalnej zawartości BX
   POP   BX        ; zapisanie do BX oryginalnej zawartości AX – zamiana miejscami
RET
END
```

**Zadanie**

Zapisać na stosie liczby od 1 do 12, a następnie pobrać je ze stosu i wpisać do tablicy TAB.

**Uwaga**: Wykorzystanie rejestru BP do adresowania stosu będzie omówione przy omawianiu procedur.

## Procedury

Procedura jest fragmentem kodu wywoływanego z program w celu realizacji określonego zadania. Zastosowanie procedur sprawia, że program uzyskuje określoną strukturę i jest łatwiejszy do zrozumienia. Wykorzystanie procedur umożliwia powtórne użycie kodu (wielokrotne wykorzystanie procedur w różnych programach). Użycie procedur zmniejsza wielkość pamięci zajmowanej przez program.

Po zakończeniu procedury program z reguły wraca do miejsca wywołania (następnej instrukcji).

Składnia procedury jest następująca:

```
name  PROC

    ; kod procedury ...

RET
name  ENDP
```

name – nazwa procedury.

Instrukcja **RET** kończy wykonanie procedury i nakazuje powrót do program wywołującego.

**PROC** oraz **ENDP** są dyrektywami kompilatora i nie generują żadnego kodu. Kompilator po prostu zapamiętuje adres procedury.

Instrukcja **CALL** służy do wywołania procedury.

```
CALL  name
```

Oto przykład:

```
ORG   100h
  MOV   AX, 1

  CALL   m1

  MOV    AX, 2

  CALL   m1

  RET              ; powrót do system operacyjnego.

  m1    PROC
    ADD   AX, 5
    RET            ; powrót do program wywołującego.
  m1    ENDP

END
```

Istnieją różne **sposoby przekazywania parametrów i wyniku** działania pomiędzy programem wywołującym a funkcję. Najprostszym jest **przekazywanie przez rejestry**.

**Przykład:**

W następującym przykładzie procedura otrzymuje dwa parametry przez rejestry **AL** oraz **BL**, mnoży je i wynik zwraca przez rejestr **AX**:

```
ORG   100h
  MOV AH, 0
  MOV   AL, 1
  MOV   BL, 2

  CALL  m2
  CALL  m2
  CALL  m2
  CALL  m2

RET             ; powrót do system operacyjnego.

m2    PROC
  MUL   BL          ; AX = AL * BL.
  RET             ; powrót do program wywołującego.
m2    ENDP

END
```

W powyższym przykładzie obliczona jest wartość **2** do potęgi **4.**

**Zadania**.

1. Napisać procedurę, która mały znak ASCII zamienia na duży i wykorzystać ją w programie. Inne znaki pozostawić bez zmiany.
2. Napisać procedurę, która małe znaki łańcucha (zakończonego 0) zamienia na duże. Użyć ją w programie, który wczytuje łańcuch z klawiatury, zamienia na duże litery i wyświetla na ekranie. Dokonywać zamiany Enter na 0 i odwrotnie.

**Przykład:**

W następnym przykładzie używana jest procedura do wyświetlenia komunikatu *Hello World*:

```
ORG    100h

LEA    SI, komunikat        ; pobranie adresu zmiennej  komunikat do SI.

CALL   drukuj

RET                 ; powrót do systemu operacyjnego.

; ========================================================
; procedura wyświetla napis zakończony zerem (znacznik końca)
; adres łańcucha znaków przekazywany jest przez rejestr SI:
drukuj     PROC

next_char:
   CMP  b.[SI], 0               ; sprawdzenie czy osiągnięto znacznik końca (0)
   JE   stop            ; jeśli tak to koniec procedury

   MOV  AL, [SI]     ; pobranie następnego kodu ASCII .

   MOV  AH, 0Eh      ; funkcja systemowa.
   INT  10h          ; wyświetla znak zapisany w  AL.

   INC  SI

   JMP  next_char    ; przejście do następnego znaku.

stop:
RET                 ; powrót do program wywołującego.
drukuj     ENDP

; ========================================================

komunikat    DB  'Hello World!', 0   ; tablica znaków zakończona znacznikiem końca (0).

END
```

Prefiks "**b.**" przed [SI] oznacza, że porównujemy bajty; porównując wartości 16-bitowe używamy prefiksu "**w.**" . Jeśli porównywana jest zawartość rejestru – prefiksy te nie są potrzebne, gdyż kompilator zna rozmiary rejestrów.

**Przykład definicji i użycia procedury:**

```
outhex   PROC
;specyfikacja funkcji
; wejście:  BH zawiera  bajt
; wyjście:   pisz szesnastkową  wartość bajtu
;zapamiętanie stanu procesora (rejestrów wykorzystywanych przez funkcję)
push  AX
 push  BX
 push  CX
 push  DX

;operacje realizowane przez funkcję

mov   DL, BH
 mov  CL, 4
 shr    DL, CL      ; analiza 4 bitów mniej znaczących
 mov  CX, 2    ; pętla 2 razy
 AGAIN:
       cmp  DL, 9
        ja  BIG
        add  DL, 30h
        jmp  PRINT
        BIG:   add  DL, 37h
        PRINT:  mov  AH, 2
        int  21h   ; wyświetlenie znaku

        mov  DL, BH
        and  DL, 0Fh    ; wyzeruj starsze   4 bity
  loop  AGAIN

;odtworzenie stanu procesora (w odwrotnej kolejności!!!)
pop DX
 pop CX
 pop BX
 pop AX
;powrót do program wywołującego
 ret
outhex  ENDP
```

---

**Użycie procedury**

```
mov  BH, zmienna      ; podaj argument dla OUTHEX

 call   outhex
```

**Przykład procedury komunikującej się z programem głównym za pośrednictwem stosu:**

```
N  EQU  4
org  100h
; sumowanie kilku liczb przekazywanych przez stos
push  a
push  b
push  c
push  d
push  N                 ; liczba sumowanych elementów
call   suma

add   SP,10     ; przesunięcie wskaźnika stosu o obszar zajęty przez parametry

ret
suma  PROC
    mov  BP, SP         ; ustawienie rejestru bazowego dla dostępu do parametrów
    push CX
    mov  CX, [BP+2]
    mov  AX, 0
    mov  SI, 4
    ets:  add AX, [BP+SI]
       add  SI, 2
    loop  ets
    pop  CX
    ret             ; wynik zwracany przez AX
suma  ENDP

a   dw  3
b   dw  2
c   dw  7
d   dw  4
end
```

Narysować obraz stosu w momencie wywołania procedury i prześledzić operacje na stosie podczas pracy programu.

## Wykorzystanie stosu do komunikacji z procedurą.

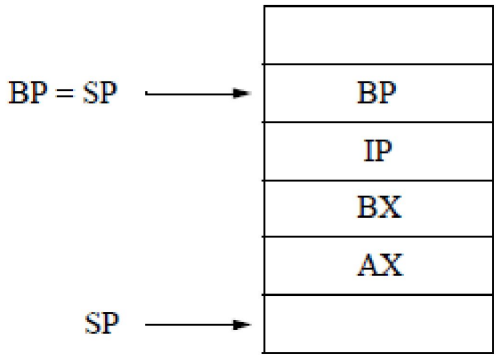
```
;Program procSuma.asm
;dodaj_proc.asm
;-------------------------------------------------------------
org 100h

  mov  AH, 09H          ;wyświetlenie komunikatu
  lea  DX, komunikat
  int  21H ;dos interrupt
  ; wczytywanie wartości z klawiatury
  mov  AH, 0AH          ;buforowane wejście z klawiatury
  lea  DX, listaPar
  int  21H
 ; pobierz dwie liczby całkowite
  mov  AL, poleOper   ;pobierz 1 znak z poleOper
  mov  AH, 00H
  push  AX              ; 1 argument na stos
  mov   BL, poleOper +1 ; pobierz 2 znak z poleOper
  mov   BH, 00H
  push  BX              ; 2 argument na stos
  call   oblicz         ;odłożenie adresu powrotu na stosie i przejście do procedury

  ;miejsce powrotu po wykonaniu procedury
  ;zapisanie sumy do zmiennej wynik
  mov  wynik+8, AH
  mov  wynik+9, AL
  ;wyświetlenie wyniku
  mov  AH, 09H ;wyświetlenie napisu
  lea  DX, wynik
  int  21H
ret
;-------------------------------------------------------------
oblicz  PROC
        push  BP              ;wykorzystany będzie rejestr BP
        mov   BP, SP          ;BP wskazuje na wierzchołek stosu
        mov   AX, [BP+4]      ;pobranie pierwszego operandu
        add    AX, [BP+6]     ;dodanie drugiego argumentu do AX
        aaa                   ;korekcja dziesiętna dla kodów ASCII
        or    AX, 3030H        ;zamiana na kod ASCII
        pop  BP               ;odtworzenie BP
        ret   4           ;pobranie  IP z dodaniem  4 do SP tak by wskazywał początkową lokalizację
oblicz  ENDP

listaPar  LABEL BYTE
maxNapis  DB 5
liczbaZn  DB ?
```

```
poleOper  DB 5 DUP(?)
komunikat  DB  0DH, 0AH, ”Podaj dwie liczby jednocyfrowe: $”
wynik  DB  0DH, 0AH, ”Suma =          $”
;-------------------------------------------------------
END
```




**Wykorzystanie stosu w powyższym programie**

**Zadania**

1. Napisz procedurę obliczającą wartość f(x)=0 dla x<0 oraz f(x)=$x^2$ dla x≥0. Wykorzystaj ją w programie.
2. Napisz procedurę, która zwraca wartość największej z trzech przekazywanych jej liczb całkowitych. Wykorzystaj ją w programie głównym.
3. Napisz procedurę, która zwraca wartość największą (najmniejszą, sumę elementów) z tablicy liczb całkowitych. Wykorzystaj ją w programie głównym.
4. Napisz procedurę obliczającą wartość f(x)= -1 dla x<0, f(x)= 0 dla x=0 oraz f(x)=+1 dla x>0. Wykorzystaj ją w programie.
5. Napisać procedurę, która otrzymuje jako parametry łańcuch S oraz liczbę powtórzeń N, a następnie wyświetla na ekranie łańcuch S N-razy przy czym za każdym razem poprzedza go liczbą spacji o 1 większą (aż do N-1 spacji).