

1. Przykładowe wykorzystanie stosu.

Stos jest rodzajem pamięci, a dokładniej mówiąc segmentu pamięci, w którym można szybko "odłożyć" dane, które potem można od razu wykorzystać. Stos działa na zasadzie LIFO - Last In First Out, tzn. ostatnia odłożona wartość będzie pierwszą pobraną. Ponadto wartość taka jest "usuwana" (przesuwany jest wskaźnik sp) więc można ją pobrać tylko raz. Stos został dokładnie opisany w części teoretycznej. W Assemblerze rejestr ss wskazuje na początek segmentu stosu.

Poniższy program modyfikuje przykład z części "Podstawowa tablica" - tutaj wartości tablicy powiększone o 1 zapisywane są na stosie, aby potem mogły zostać "ściągnięte" i wyświetlone bez modyfikacji tablicy. Stos może zapisywać tylko wartości 16-bitowe.

```
.Model small // Inicjalizacja wielkości modelu - small.
```

```
.stack 100h // Inicjalizacja stosu o wielkości 100h. W przeciwieństwie do rejestru ds - segment stosu od razu zapisuje  
swoj adres w rejestrze ss (tak samo kod w rejestrze cs)
```

```
.data // Inicjalizacja segmentu danych.
```

```
array db 1,4,4,5,2,7,8 //definicja zmiennej array jako tablicy - inicjalizacja wartościami liczbowymi
```

```
array_size dw 7 //definicja zmiennej array_size i inicjalizacja - rozmiar tablicy
```

```
message db 13, 10, "Array Values (reverse) increment by 1: $" /// Definicja zmiennej message typu db. Inicjalizacja  
wartościami: 13, 10 -> kod nowego znaku, oraz łańcuch  
znaków zakończony $.
```

```
.code // Inicjalizacja segmentu kodu.
```

```
MAIN PROC // Inicjalizacja głównej procedury.
```

```
mov ax,@data //Skopiowanie adresu segmentu danych w pamięci (przydzielonym przez procesor w lini .data) do  
rejestru ax. Rejestr ds (wykorzystany w następnej linii) przyjmuje tylko wartości z innych rejestrów  
ogólnego przeznaczenia, zatem wykorzystano rejestr AX.
```

```
mov ds,ax // Skopiowanie wartości rejestru ax do rejestru ds. Teraz rejestr ds wskazuje na początek segmentu  
danych w pamięci, co umożliwi i ułatwi programiście wiele operacji na pamięci.
```

```
xor cx, cx //Wyczyszczenie rejestru cx
```

```
mov cx, array_size // Skopiowanie wartości w pamięci, na którą wskazuje zmienna array_size do rejestru cx.
```

```
xor di, di //Wyczyszczenie rejestru di. Rejestr ten wykorzystywany jest często jako indeks tablicy.
```

```
mov bp, sp //Skopiowanie początkowej wartości wskaźnika wierzchołka stosu (rejest sp) do rejestru bp (wskaźnik do  
danych w stosie). Ten zabieg pozwala zapamiętać adres pierwszego (czyli zarazem ostatniego na stosie)  
elementu, co dalej pozwala na wykonanie pętli ściągania ze stosu odpowiednią ilość razy (wartość  
wskaźnika sp rośnie/maleje podczas instrukcji push/pop)
```

```
loop_push: //Etykieta początku pętli - dodanie danych z tablicy (zwiększonych o 1) do stosu
```

```
xor ax, ax //Wyczyszczenie rejestru cx
```

```
mov al, array[di] //Skopiowanie wartości tablicy o indeksie di do rejestru dl.
```

inc ax //Zwiększenie wartości rejestru ax o 1.

push ax //Skopiowanie wartości rejestru ax do pamięci w segmencie stosu na wierzchołek wskazujący przez wartość rejestru sp (adres rzeczywisty to ss:sp). Zmniejszenie wartości rejestru sp.

inc di //Zwiększenie wartości rejestru di o 1.

loop loop_push //Jeśli wartość rejestru cx jest większa od 0 - skocz do etykiety start i zmniejsz wartości cx o 1 (dec cx). Jeśli cx równe jest 0, wyonaj program do następnej lini.

mov ah, 09h //Inicjalizacja przerwania int 21h - instrukcja 09 - wypisanie łańcucha znaków z pamięci.

mov dx, offset message //Skopiowanie wartości offsetu zmiennej message do rejestru dx.

int 21h // Wykonanie przerwania int 21h - instrukcji 09

loop_inc: //Etykieta początku pętli - ściągnięcie danych ze stosu i wyświetlenie w konsoli

xor ax, ax //Wyczyszczenie rejestru cx

pop ax //Skopiowanie wartości z pamięci z wierzchołka wskazanego przez wartość rejestru sp (adres rzeczywisty to ss:sp) do rejestru ax. Zwiększenie wartości rejestru sp.

mov ah, 02h //Inicjalizacja przerwania int 21h - instrukcja 02.

mov dl, al //Skopiowanie wartości rejestru al do rejestru dl

add dl, 30h //Konwersja wartości liczbowej rejestru dl do ASCII.

int 21h // Wykonanie przerwania int 21h - instrukcji 02 - wyświetlenie wartości rejestru dl w konsoli.

mov ah, 02h //Inicjalizacja przerwania int 21h - instrukcja 02.

mov dl, " " //Skopiowanie symbolu spacji (ASCII) do rejestru dl.

int 21h // Wykonanie przerwania int 21h - instrukcji 02 - wyświetlenie wartości rejestru dl w konsoli.

cmp sp, bp //Porównanie wartości rejestru sp do wartości rejestru bp, jeśli są równe - stos jest pusty, a wartość instrukcji równa jest 0 - flaga ZF => 1.

jnz loop_inc //Skocz do loop_inc jeśli flaga ZF jest równa 0, czyli wartości porównane w cmp były różne -> Stos nie jest jeszcze pusty.

mov ah, 4Ch //Inicjalizacja przerwania int 21h - instrukcja 4Ch - zakończenie programu

int 21h // Wykonanie przerwania int 21h - instrukcja 4Ch - zakończenie programu

main endp // Koniec głównej procedury.

end main // Koniec pliku kodu.