Kamil Skarżyński

Programowanie niskopoziomowe
Typy danych, operacje przesyłania danych
Laboratorium 03

1. Wprowadzenie

Podczas laboratoriów zapoznamy się z:
1. Zczytywaniem danych z konsoli,
2. Sposobem alokacji pamięci dla tablic danych,
3. Sposobem na uzyskanie adresu zmiennej w pamięci,
4. Dyrektywami pomagającymi określić rozmiar zmiennej,
5. Pętlami,
6. Rejestrami oraz instrukcjami wspomagającymi przesyłanie danych

2. Zadania

1. Utwórz tablicę bajtów o nazwie "source" i wyświetl zachęte dla użytkownika by wprowadził ciąg 10 znaków który zostanie zapisany do tablicy(3.7). Następnie skopiuj ją do drugiej tablicy o nazwie "destination". Nie wykorzystuj instrukcji MOVSB/W/D. Wyświetl na konsoli zawartość drugiej tablicy (3 punkty).
2. Wkonaj poprzednie zadanie, lecz z wykorzystaniem instrukcji MOVSB oraz rep. Nie wykorzystuj pętli (1 punkt).
3. Wprowadz za pomocą konsoli 10 znaków do tablicy "zrodlo" a następnie przenieś je do tablicy "przeznaczenie" w odwrotnej kolejnosci (pomoc: 3.9.5, 3.9.6) wyświetl wynik na konsoli (2 punkty),
4. Stwórz dwie tablice podwójnych słow o rozmiarze 30 różniące się tylko jednym elementem, sprawdz który element jest różny. Nie wykorzystuj pętli. Wykorzystaj instrukcję CMPSB. Wypisz na konsoli indeks znaku który jest inny (2 punkty).
5. Wprowadz kolejno 10 liczb(decymalnie) za pośrednictwem konsoli. Oblicz ich sumę i wypisz wynik (1 punkt).

3. Pomoc

3.1. Alokacja pamięci

Alokacja jednego bajtu:

```
zmiennaBajtowa DB 0
zmiennaBajtowa2 BYTE 0
```

Alokacja dwóch bajtów:

```
zmiennaDwuBajtowa DW 0
zmiennaDwuBajtowa2 WORD 0
```

Alokacja czterech bajtów:

```
zmiennaCzteroBajtowa dd 0
zmiennaCzteroBajtowa2 DWORD 0
```

Alokacja tablicy bajtów o rozmiarze 10 wypełnionej 0:

```
tablicaBajtow DB 10 dup(0)
```

Alokacja tablicy słów o rozmiarze 5 wypełnionej 5:

```
tablicaSlow DW 5 dup(5)
```

Alokacja tablicy podwójnych słów o rozmiarze kodem ascii znaku A:

```
tablicaPodwojnychSlow DD 8 dup("A")
```

Alokacja tablicy 10 bajtów z konkretnymi wartościami:

```
tablicaBajtow DB 1,2,3,4,5,6,7,8,9,0
```

Alokacja tablicy 10 bajtów wartościami z tabeli ascii:

```
tablicaBajtow DB "1234567890"
```

3.2. Uzyskanie adresu zmiennej w pamięci

W celu uzyskania adresu zmiennej w pamięci możemy użyć operatora OFFSET. Zwraca on liczbę bajtów od początku segmentu do danej etykiety. W trybie chronionym rozmiar adresu wynosi 32 bity.

Dla przykładu zakładając że na segment danych przeznaczone zostały adresy od 00404000h to:

```
.data
bVal BYTE ?
wVal WORD ?
dVal DWORD ?
dVal2 DWORD ?
.code
mov ESI,OFFSET bVal ; ESI = 00404000
mov ESI,OFFSET wVal ; ESI = 00404001
mov ESI,OFFSET dVal ; ESI = 00404003
mov ESI,OFFSET dVal2 ; ESI = 00404007
```

3.3. Określenie rozmiaru zmiennej

W tym celu możemy wykorzystać 3 operatory: TYPE, LENGTHOF, SIZEOF

TYPE - zwraca rozmiar typu danych w bajtach. Dla BYTE jest to 1, WORD jest to 2, DWORD jest to 4. LEGTHOF - zwraca ilość elementów, w przypadku jeśli pod zmienną nie kryje się tablica będzie to 1. SIZEOF == TYPE * LENGTHOF. Pozwala to na obliczenie ile bajtów zajmuje tablica.

Przykłady:

```
mov EAX, TYPE zmiennaBajtowa  ;EAX == 1
mov EAX, TYPE zmiennaDwuBajtowa  ;EAX == 2
mov EAX, TYPE zmiennaCzteroBajtowa  ;EAX == 4
mov EAX, TYPE tablica10Slow       ;EAX == 2

mov EAX, LENGTHOF zmiennaCzteroBajtowa   ;EAX == 1
mov EAX, LENGTHOF zmiennaCzteroBajtowa   ;EAX == 1
mov EAX, LENGTHOF zmiennaCzteroBajtowa   ;EAX == 1
mov EAX, LENGTHOF tablica10Slow  ;EAX == 10

mov EAX, SIZEOF zmiennaBajtowa   ;EAX == 1*1 = 1
mov EAX, SIZEOF zmiennaDwuBajtowa          ;EAX == 2*1 = 2
mov EAX, SIZEOF zmiennaCzteroBajtowa       ;EAX == 4*1 = 4
mov EAX, SIZEOF tablica10Slow     ;EAX == 2*10 = 20
```

3.4. GetStdHandle - pobranie uchwytu do konsoli

W celu uzyskania uchwytu do aktualnie uruchomionego okna konsoli możemy wykorzystać procedure GetStdHandle:

```
HANDLE WINAPI GetStdHandle(
_In_ DWORD nStdHandle
);
```

nStdHandle może przyjmować następujące wartości:

1. -10 - uchwyt wejściowy, do odczytu z konsoli,
2. -11 - uchwyt wyjsciowy, do wypisywania znaków na konsolę,
3. -12 - uchwyt umożliwiający odczyt błędów.

Uchwyt zwracany jest do rejestru EAX po wykonaniu procedury.

Przykład wykorzystania:

```
push STD_OUTPUT_HANDLE ;stala -11
call GetStdHandle
mov outputHandle, EAX
```

3.5. WriteConsoleA - wypisywanie znaków na konsolę

Biblioteki Win32 udostępniają procedury umożliwiające odczyt znaków z interesującej nas konsoli. Można wykorzystać do tego procedurę WriteConsoleA o następującej sygnaturze:

```
BOOL WINAPI WriteConsoleA(
_In_             HANDLE  hConsoleOutput,
_In_       const VOID    *lpBuffer,
_In_             DWORD   nNumberOfCharsToWrite,
_Out_            LPDWORD lpNumberOfCharsWritten,
_Reserved_       LPVOID  lpReserved
);
```

Parametry:

1. hConsoleOutput - uchwyt wyjściowy do konsoli, pobierany za pomocą procedury GetStdHandle (3.4),
2. *lpBuffer - adress tablicy przechowującej znaki do wypisania,
3. nNumerOfCharsToWrite - liczba znaków które zostaną wypisane z poprzednio podanego adresu,
4. lpNumberofCharsWriten - adres zmiennej typu DWORD do której procedura zapisze ilość faktycznie wypisanych znaków,
5. lpReserved - wstawiamy null czyli 0

Przykład wykorzystania:

```
push 0
push OFFSET nOfCharsWritten
push nOfCharsToWrite
push OFFSET charsToWrite
push outputHandle
call WriteConsoleA
```

Segment danych:

```
nOfCharsWritten DWORD 0
charsToWrite    BYTE "Wprowadz argument A",0
nOfCharsToWrite DWORD $ - charsToWrite
```

3.6. CharToOemA - konwersja znaków

Konsola systemu windows wykorzystuje tak naprawdę kodowanie OEM, więcej można znaleźć tutaj. W tym celu by poprawnie wyświetlić polskie znaki, musimy przekonwertować znaki ASCII na OEM za pomocą procedury CharToOemA:

```
BOOL WINAPI CharToOemA(
_In_  LPCTSTR lpszSrc,
_Out_ LPSTR   lpszDst
);
```

Parametry:

1. lpszSrc - adres tablicy z znakami którą chcemy przekonwertować,
2. lpszDst - adress tablicy do której chcemy zapisać przekonwertowane znaki,

Przykład wykorzystania:

```
push OFFSET charsToWrite
push OFFSET charsToWrite
call CharToOemA
```

Jako adres docelowy możemy wykorzystać naszą tablicę z znakami, ponieważ pierwotne kodowanie nie będzie nam potrzebne.

3.7. ReadConsoleA - odczytywanie znaków z konsoli

W celu wprowadzania znaków przez konsolę możemy wykorzystać procedure ReadConsoleA o następującej sygnaturze:

```
BOOL WINAPI ReadConsole(
_In_      HANDLE  hConsoleInput,
_Out_     LPVOID  lpBuffer,
_In_      DWORD   nNumberOfCharsToRead,
_Out_     LPDWORD lpNumberOfCharsRead,
_In_opt_  LPVOID  pInputControl
);
```

Parametry:
1. hConsoleInput - uchwyt wejściowy do konsoli, pobierany za pomocą procedury GetStdHandle(3.4),
2. lpBuffer - adress tablicy do której zapisane zostaną odczytane znaki,
3. nNumberOfCharsToRead - liczba znaków która ma zostać odczytana z konsoli,
4. lpNumberOfCharsRead - adres do miejsca pamięci do którego zapisana zostanie liczba zczytanych znaków,
5. pInputControl - wstawiamy null czyli 0

Przykład wykorzystania:

```
push 0
push OFFSET nOfCharsRead
push 10
push OFFSET inputBuffer
push inputHandle
call ReadConsoleA

mov EBX, OFFSET inputBuffer
add EBX, nOfCharsRead
mov [EBX-2], BYTE PTR 0
```

W celu zrozumienia 3 ostatnich instrukcji wykorzystaj debugger i podejrzyj co zapisywane jest w inputBuffer. Na potrzeby procedury konwertującej znaki na liczbę, nasz ciąg znaków musi kończyć się nullem (00 w pamięci).

3.8. Pętle

Najłatwiejszym sposobem organizacji pętli jest skorzystanie z instrukcji loop. Odejmuje ona od rejestru ECX (licznik) wartość 1, następnie sprawdza czy rejestr ECX ma wartość 0, jeżeli nie to wykonywany jest skok do etykiety podanej jako parametr.

Gdy programujemy w asemblerze musimy zadbać o ty aby wewnątrz pętli wartość rejestru ECX nie uległa zmianie. Dobrym nawykiem jest odkładanie wartości rejestru ECX na stos na początku pętli i pobierani tej wartości do licznika przed użyciem instrukcji LOOP na końcu pętli.

Przykład:

```
mov ECX, liczbaWykonan
petla:
push ECX
;operacje wymagajace petli
pop ECX
LOOP petla
```

3.9. Operacje łańcuchowe

W operacjach łańcuchowych wykorzystujemy rejestry indeksowe ESI (Source Index) i EDI (Destination Index), których wartość automatycznie zwiększa (lub zmniejsza w zależności od flagi kierunku).

3.9.1. MOVSB, MOVSW, MOVSD

Rozkazy MOVSB, MOVSW i MOVSD kopiują dane, z pod adresu wskazanego przez ESI do pamięci wskazanej przez adres w rejestrze EDI. Po wykonaniu jednego z wyżej wymienionych rozkazów, rejestry ESI i ESI są automatyucznie zwiększane/zmniejszane o:

1. MOVSB zwiększane/zmniejszane o 1
2. MOVSw zwiększane/zmniejszane o 2
3. MOVSD zwiększane/zmniejszane o 4

Przykład skopiowania danych z zmiennej "zrodlo" do zmiennej "przeznaczenie" (zmienne są typu DWORD):

```
mov ESI, OFFSET zrodlo
mov EDI, OFFSET przeznaczenie
movsd
```

By zmienić flagę kierunku możemy wykorzystać DF (Direction Flag) gdy jest ustawiona na 0, rejestry ESI i EDI są zwiększane, gdy na 1 są zmniejszane.
Przykład manipulacji DF:

```
CLD ;wyczyszczenie DF na 0
STD      ;ustawienie DF na 1
```

3.9.2. Prefiks REP

Prefiks REP może zostać wstawiony przed rozkaz MOVSB, MOVSW lub MOVSD. Rejestr ECX określa liczbę powtórzeń.

Przykład:

```
.data
zrodlo DWORD 20 DUP(?)
przeznaczenie DWORD 20 DUP(?)
.code
main proc
CLD ;wyczyszczenie DF na 0
mov ECX, LENGTHOF zrodlo ; ustawienie licznika dla prefixu REP
mov ESI, OFFSET zrodlo
mov EDI, OFFSET przeznaczenie
```

```
rep movsd
main endp
```

3.9.3. CMPSB, CMPSW i CMPSD

Rozkazy CMPSB,CMPSW i CMPSD porównują operand wskazany przez ESI z operandem wskazanym przez EDI.

1. CMPSB proównuje bajty,
2. CMPSW porównuje słowa,
3. CMPSD prównuje podwójne słowa.

Rozkazy te często używane są w poołączeniu z prefixami:

1. REPE - Rep if Equal
2. REPNE - Rep if Not Equal

Prefiks REPE możemy używać np. do porównania dwóch ciągów znaków (stringów).

Przykład:

```
.data
zrodlo DWORD 20 DUP(0)
przeznaczenie DWORD 20 DUP(0)
.code
main proc
CLD ;wyczyszczenie DF na 0
mov ECX, LENGTHOF zrodlo ; ustawienie licznika dla prefixu REP
mov ESI, OFFSET zrodlo
mov EDI, OFFSET przeznaczenie
repe cmpsd

push 0
call ExitProcess
main endp
```

3.9.4. SCASB, SCASW, SCASD

Porównuje wartości w AL/AX/EAX z odpowiednio bajtem/słowem/podwójnym słowem wskazywanym przez rejestr EDI. Użyteczne w przypadku poszukiwania konkretnego elementu w tablicy lub szukaniu elementu nie pasującego do danego wzorca.

3.9.5. STOSB, STOSW, STOSD

Zapisują wartości odpowiednio z rejestrów AL/AX/EAX do miejsca w pamięci wskazywanego przez EDI, oraz zwiększa ten rejestr o 1/2/4. Może być wykorzystywane do np wy wypełniania tablicy.

3.9.6. LODSB, LODSW, LODSD

Ładują wartość z pamięci o adresie wskazywanym przez ESI do rejestrów AL/AX/EAX, oraz zwiększa odpowiednio rejestr ESI o 1/2/4.

Przykład:

```
.data
        tabl DWORD "ABCDEF",0
.code
main proc
```

```
mov ESI, OFFSET tabl ;adres tabl w ESI
mov EAX, 0 ;wyzerowanie AL

lodsb ;zaladowanie do rejestru AL kodu ASCII znaku A
lodsb ;zaladowanie do rejestru AL kodu ASCII znaku B

push 0
call ExitProcess
main endp
```