

## Pamięć

Umieszczamy czasem jakieś dane w pamięci, może zwróciliście uwagę. Powiem więcej, umieszczany tam też jest cały program, który będziemy wykonywać. Możemy to zaobserwować chociażby wyświetlając sobie jakiś program przy użyciu najzabawniejszego programu na świecie, xxd. W górnej części screena obraz z insighta, w dolnej z xxd.

```
0025:0100  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF  .....!..L.!0000
0025:0110  BA 0C 01 BB 1E 0C 01 BB 00 4C CD 21 41 53 45 4D  ||001^001.L=!ASEM  00
0025:0120  42 4C 45 52 20 4A 45 53 54 20 54 41 4B 49 20 53  BLER JEST TAKI S  Z1.BIN
0025:0130  55 50 45 52 24 00 00 00 00 00 00 00 00 00 00 00  UPER$.
0025:0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

michal@michal-virtualbox:~/dosbox/WIA2/LAB5$ xxd Z1.BIN
00000000: ba0c 018b 1e0c 01b8 004c cd21 4153 454d  .....L.!ASEM
00000010: 424c 4552 204a 4553 5420 5441 4b49 2053  BLER JEST TAKI S
00000020: 5550 4552 2400  UPER$.
```

Co tu dokładnie widzimy? Zassemblewano do formy pliku binarnego kod asemblera. Sam program na końcu ma zadeklarowany pod etykietą „string” ciąg znaków „ASSEMBLER JEST TAKI SUPER\$”. Pewną różnicę możemy zaobserwować przy adresach w pamięci. W dolnej części program zaczyna się od 0000, natomiast w dosie już od 0100. Jest tak dlatego, że użyliśmy globalnego offsetu, który wszystko wczytuje nam zaczynając od zadanego adresu. W tym wypadku używamy org 100h, a za tym wszystko przesunięte jest o 100h „komórek”.

Założmy następujący kod:

```
Org 100h
mov DX, string
mov BL, [string]

mov AX, 4c00h
int 21h

string db "ASSEMBLER JEST TAKI SUPER$"
```

On co prawda nie da nam żadnej informacji zwrotnej na terminalu, ale pozwoli pokazać, jak konkretnie dostajemy się do pamięci. Po wrzuceniu takiego programu do insighta możemy zobaczyć:

```

DOSBox 0.74-3, Cpu speed: ...meskip 0, Program: INSIGHT - x
80486 Insight 1.24
0100 BA0C01 mov dx,010C
0103 8A1E0C01 mov bl,[010C]
0107 BB004C mov ax,4C00
010A CD21 int 21
010C 41 inc cx
010D 53 push bx
010E 45 inc bp
010F 4D dec bp
0110 42 inc dx
0111 4C dec sp
0112 45 inc bp
0113 52 push dx
0114 204A45 and [bp+si+45],cl
0117 53 push bx
0118 54 push sp
0119 205441 and [si+41],dl
011C 4B dec bx
0D25:0107
0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0D25:0100 BA 0C 01 BA 1E 0C 01 B8 00 4C CD 21 41 53 45 4D ||90E0A90E7.L=1ASEM
0D25:0110 42 4C 45 52 20 4A 45 53 54 20 54 41 4B 49 20 53 BLER JEST TAKI S
0D25:0120 55 50 45 52 24 00 00 00 00 00 00 00 00 00 00 00 UPER$.....
0D25:0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0D25:0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
AX=0000 SI=0100 CS=0D25
BX=0041 DI=FFFE DS=0D25
CX=00FF BP=091C ES=0D25
DX=010C SP=FFFE SS=0D25
IP=0107 Flags=7202
OF DF IF SF ZF AF PF CF
0 0 1 0 0 0 0 0
SS:000C 0194 +000A 0000
SS:000A 0BA1 +0008 0000
SS:0008 DEAD +0006 0000
SS:0006 FFFF +0004 0000
SS:0004 EA00 +0002 0000
SS:0002 9FFF +0000 0000
SS:0000 20CD -0002 0000
SS:FFFE 0000 -0004 0000

```

Pierwsza instrukcja

### MOV DX, string

Przenosi do rejestru DX adres etykiety, czyli początku naszego stringa – 010C. Natomiast

### MOV BL, [string]

Przenosi bezpośrednio do rejestru BL zawartość wskazywaną przez zadany adres. W pamięci pod adresem 010C znajduje się literka A, tak więc do rejestru BL przeniesiona została wartość 'A' w ASCII, czyli 41h. W taki sam sposób możemy również ustalić, co znajduje się np. na czwartym miejscu stringa

### MOV BL, [string+4]

Zadanie 1

Za pomocą przerwania 21h/AH=0Ah wczytaj ciąg znaków, a następnie wydrukuj trzeci znak tego stringa. Do druku można użyć teletype output (int10h/AH=0Eh). Przypominam, że string zbuforowany na początku umieszcza jeszcze maksymalny rozmiar bufora oraz liczbę znaków wprowadzoną, więc trzeba to przewidzieć i dodać w odpowiednim miejscu o 2 więcej.

Proponuję zadeklarować sobie jakąś pustą etykietę i przypisać jej adres do DX, żebyśmy mieli do niej potem dostęp. Np.:

```
string times 10 db 0
```

zaalokuje nam miejsce na 10 znaków, tymczasowo zapełniając je zerami.

Aby uniknąć konfuzji przy drukowaniu proponuję również wydrukować sobie newline po wczytaniu, a przed wydrukiem. Trust me

```
MOV AH, 0Eh
MOV AL, 0Ah
INT 10h
MOV AL, 0Dh
INT 10h
```

Jak coś w ogóle umieścić w pamięci?

Generalnie przed chwilą wczytaliśmy sobie coś do pamięci, ale za pomocą przerwania. Jednak chcielibyśmy umieścić w pamięci jakąś konkretną wartość i tu pojawia się problem, aczkolwiek niewielki. Czasami po prostu musimy pomóc asemblerowi z tłumaczeniem odniesień do pamięci. Dla przykładu: chcielibyśmy zapisać '\$' pod adresem 010C. Instrukcja:

```
MOV [010ch], '$'
```

Wyrzuci nam błąd. Asembler nie wie, w jakim rozmiarze ma przenieść tę daną. Czy ma to być Byte (8 bitów), word (16 bitów), czy może dword (32 bity). Musimy mu w tym pomóc definiując rozmiar operacji

```
MOV byte [010ch], '$'
```

Już takiego błędu nie wyrzuci. Teraz ważne: to, co znajduje się w ramkach pozwala na dobranie się do pamięci, ale tylko za pomocą narzędzi, które z pamięcią współpracują. Nie możemy na przykład zrobić w ten sposób:

```
MOV DX, 010ch
MOV byte [DX], '$'
```

Zamiast podawać z palca bezpośrednio adres, bo w niektórych przypadkach go możemy przecież dokładnie nie znać, zawsze możemy odnieść się do „aliasu” adresu w pamięci, czyli etykiety

```
MOV byte [string], '$'
```

## Zadanie 2

Wydrukuj stringa, może być zdefiniowany w kodzie, ale do trzeciego znaku. Przerwanie INT 21/AH=09h drukuje stringa zaczynając od umieszczonego w DX adresu aż do napotkania znaku '\$', czyli wystarczy po trzecim znaku umieścić w pamięci \$ i załatwione.

Jak już pokazałem, jakkolwiek byśmy się nie starali, nie możemy przy użyciu rejestru ogólnego przeznaczenia wskazać adresu do zapisu w pamięci. Możemy zamiast tego skorzystać z BP, czyli base pointera.

```
MOV BP, string
MOV byte [BP], '$'
```

## Zadanie 3

Napisz program, który przyjmie stringa od użytkownika (int 21h/AH=0Ah), przesunie gdzieś kursor (INT 10h/AH=02h) i wydrukuje tego stringa na terminalu. Program ma wykryć, jak długi był przyjęty od użytkownika string i postawić na jego końcu '\$'. Zwrócić należy uwagę, że przerwanie, którym wczytujemy ciąg znaków od użytkownika na drugim bajcie umieszcza liczbę wprowadzonych znaków, wyłuskać ją możemy np.

```
MOV AL, [string+1]
```

Dla przypomnienia, instrukcje arytmetyczne:

```
INC AX          ;zwiększa AX o 1
DEC AX          ;zmniejsza AX o 1
ADD AX, 10h     ;dodaje do AX 10h – można też podać rejestr
SUB AX, 10h     ;odejmuje od AX 10h – można też podać rejestr
MUL DX         ;mnoży DX przez AX
```