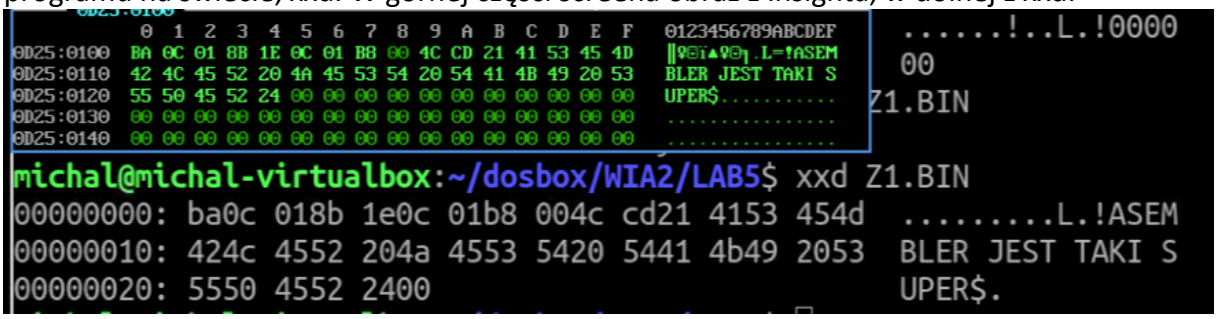
Pamięć

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.



Co tu dokładnie widzimy? Zasemblowany do formy pliku binarnego kod asemblera. Sam program na końcu ma zadeklarowany pod etykietą „string” ciąg znaków „ASEMBLER 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”.

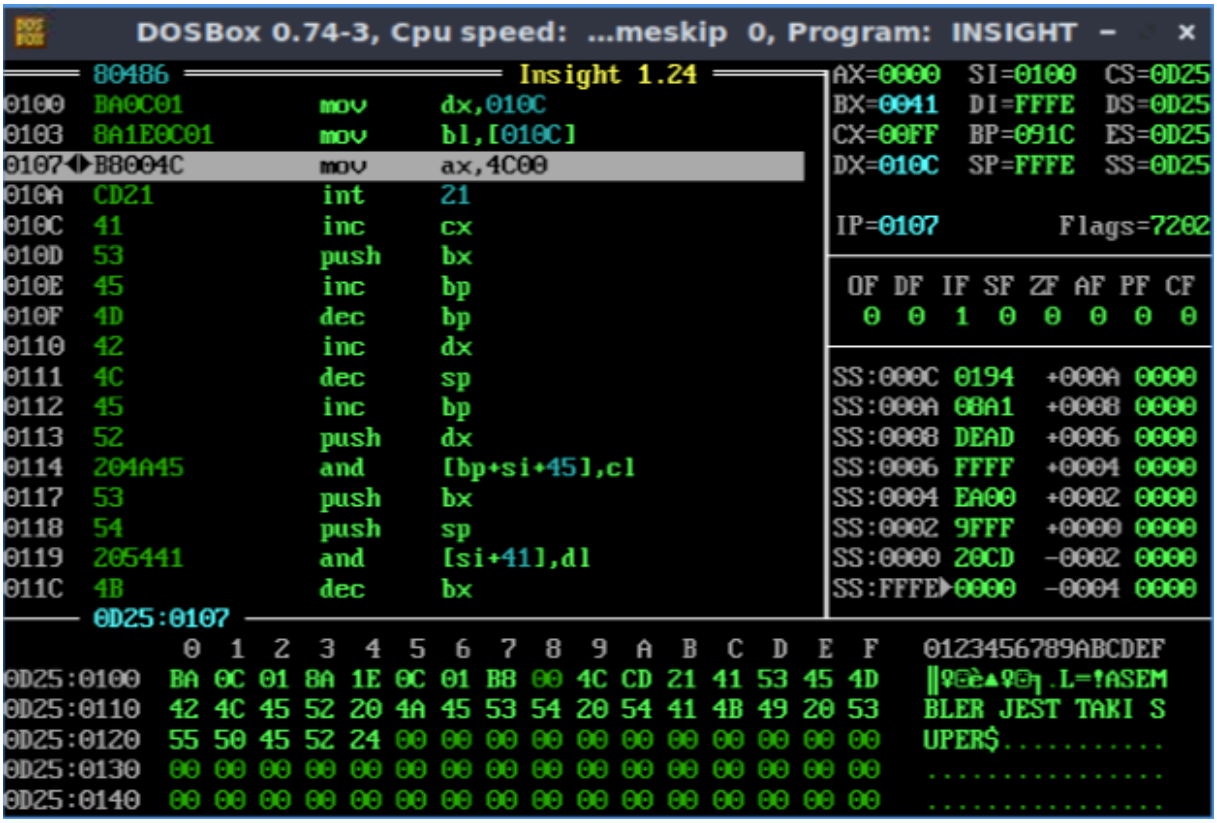
Załóżmy następujący kod:

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

mov AX, 4c00h
int 21h

string db “ASEMBLER JEST TAKI SUPER$”
```

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



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 wartoś ‘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
```